RTI Connext Modern C++ API

Version 7.7.0

1 RTI Connext **1**

1.1 Getting Started . . . 1

1.2 Available Documentation. . . . 2

1.2.1 The documents for the Core Libraries and Utilities are: . . . 2

1.2.2 The API Reference HTML documentation contains: . . . 2

1.3 Feedback and Support for this Release. . . . 2

2 Module Index **3**

2.1 Modules . . . 3

3 Namespace Index **9**

3.1 Namespace List . . . 9

4 Hierarchical Index **11**

4.1 Class Hierarchy . . . 11

5 Class Index **23**

5.1 Class List . . . 23

6 Module Documentation **43**

6.1 Clock Selection . . . 43

6.1.1 Available Clocks . . . 43

6.1.2 Clock Selection Strategy . . . 44

6.1.3 Configuring Clock Selection . . . 44

6.2 Domain Module . . . 44

6.2.1 Detailed Description . . . 45

6.3 DomainParticipants . . . 45

6.3.1 Detailed Description . . . 45

6.4 Built-in Topics . . . 46

6.4.1 Detailed Description . . . 46

6.5 Topic Module . . . 47

6.5.1 Detailed Description . . . 48

6.6 Topics . . . 48

6.6.1 Detailed Description . . . 49

6.6.2 Function Documentation . . . 49

6.6.2.1 sql_filter_name() . . . 49

6.6.2.2 stringmatch_filter_name() . . . 50

6.7 Zero Copy Transfer Over Shared Memory . . . 50

6.8 Built-in Types . . . 50

6.8.1 Detailed Description . . . 51

6.8.2 Managing Memory for Builtin Types . . . 51

6.8.3 Typecodes for Builtin Types . . . 52
6.9 Built-in Topic's Trust Types . . . 54
6.9.1 Detailed Description . . . 54
6.10 Publication Module . . . 55
6.10.1 Detailed Description . . . 55
6.11 Publishers . . . 55
6.11.1 Detailed Description . . . 56
6.12 Data Writers . . . 56
6.12.1 Detailed Description . . . 57
6.13 Flow Controllers . . . 57
6.13.1 Detailed Description . . . 58
6.13.2 Typedef Documentation . . . 58
6.13.2.1 FlowControllerSchedulingPolicy . . . 58
6.13.3 Variable Documentation . . . 58
6.13.3.1 DEFAULT_NAME . . . 58
6.13.3.2 FIXED_RATE_NAME . . . 59
6.13.3.3 ON_DEMAND_NAME . . . 60
6.14 Subscription Module . . . 61
6.14.1 Detailed Description . . . 62
6.14.2 Access to data samples . . . 62
6.14.2.1 Data access patterns . . . 62
6.15 Subscribers . . . 63
6.15.1 Detailed Description . . . 63
6.16 DataReaders . . . 63
6.16.1 Detailed Description . . . 64
6.17 Read Conditions . . . 65
6.17.1 Detailed Description . . . 65
6.18 Query Conditions . . . 65
6.18.1 Detailed Description . . . 65
6.18.2 Function Documentation . . . 66
6.18.2.1 create_query_condition_ex() [1/2] . . . 66
6.18.2.2 create_query_condition_ex() [2/2] . . . 66
6.19 Data Samples . . . 66
6.19.1 Detailed Description . . . 67
6.20 Topic Queries . . . 67
6.20.1 Detailed Description . . . 68
6.20.2 Snapshot TopicQueries . . . 69
6.20.3 Reading TopicQuery Samples . . . 69
6.20.4 Debugging Topic Queries . . . 70

6.20.4.1 The Built-in ServiceRequest DataReader . . . 70
6.20.4.2 The on_service_request_accepted DataWriter Listener Callback . . . 71
6.20.5 Typedef Documentation . . . 71
6.20.5.1 TopicQuerySelectionKind . . . 71
6.21 Sample Information . . . 71
6.21.1 Detailed Description . . . 71
6.22 Data State . . . 71
6.22.1 Detailed Description . . . 72
6.23 Infrastructure Module . . . 72
6.23.1 Detailed Description . . . 73
6.24 Multi-channel DataWriters . . . 73
6.24.1 What is a Multi-channel DataWriter? . . . 73
6.24.2 Configuration on the Writer Side . . . 74
6.24.3 Configuration on the Reader Side . . . 74
6.24.4 Reliability with Multi-Channel DataWriters . . . 74
6.24.4.1 Reliable Delivery . . . 74
6.24.4.2 Reliable Protocol Considerations . . . 74
6.25 Transports . . . 75
6.25.1 Detailed Description . . . 75
6.25.2 Overview . . . 75
6.25.3 Transport Aliases . . . 76
6.25.4 Transport Lifecycle . . . 76
6.25.5 Transport Class Attributes . . . 77
6.25.6 Transport Instance Attributes . . . 78
6.25.7 Transport Network Address . . . 79
6.25.8 Transport Send Route . . . 79
6.25.9 Transport Receive Route . . . 79
6.26 Installing Transport Plugins . . . 80
6.26.1 Loading Transport Plugins through Property QoS Policy of Domain Participant . . . 81
6.27 Built-in Transport Plugins . . . 83
6.27.1 Detailed Description . . . 83
6.28 Creating New Transport Plugins . . . 84
6.29 Queries and Filters Syntax . . . 84
6.29.1 Syntax for DDS Queries and Filters . . . 84
6.29.2 SQL grammar in BNF . . . 85
6.29.3 Token expression . . . 85
6.29.4 String Parameters . . . 87
6.29.5 Type compatability in Predicate . . . 87
6.29.6 SQL Extension: Regular Expression Matching . . . 88

6.29.7 Character Encoding . . . 89
6.29.8 Unicode Normalization . . . 89
6.29.9 Examples . . . 90
6.30 Logging and Version . . . 90
6.30.1 Detailed Description . . . 90
6.31 General Utilities and Compliance Configuration . . . 90
6.31.1 Detailed Description . . . 91
6.32 Observability Framework . . . 91
6.32.1 Detailed Description . . . 91
6.33 Request-Reply Pattern . . . 91
6.33.1 Detailed Description . . . 92
6.34 Requester . . . 92
6.34.1 Detailed Description . . . 92
6.35 Replier . . . 93
6.35.1 Detailed Description . . . 93
6.36 Queuing Pattern . . . 93
6.36.1 Detailed Description . . . 94
6.37 QueueProducer . . . 95
6.37.1 Detailed Description . . . 95
6.38 QueueConsumer . . . 95
6.38.1 Detailed Description . . . 96
6.39 QueueRequester . . . 96
6.39.1 Detailed Description . . . 96
6.40 QueueReplier . . . 96
6.40.1 Detailed Description . . . 97
6.41 Remote Procedure Call . . . 97
6.41.1 Detailed Description . . . 98
6.42 Utilities . . . 98
6.43 Durability and Persistence . . . 99
6.43.1 Durable Writer History . . . 99
6.43.2 Durable Reader State . . . 99
6.43.3 Data Durability . . . 100
6.43.4 Durability and Persistence Based on Virtual GUID . . . 100
6.43.5 Configuring Durable Writer History . . . 101
6.43.6 Configuring Durable Reader State . . . 102
6.43.7 Configuring Data Durability . . . 103
6.44 Basic QoS . . . 104
6.45 DDS Specification QoS . . . 112
6.46 DDS Extension QoS . . . 116

6.47 Functional Categories . . . 156
6.47.1 Detailed Description . . . 158
6.48 Application Acknowledgement . . . 158
6.49 Application-Defined Metadata . . . 159
6.50 Application Startup . . . 161
6.51 Batching . . . 161
6.52 Concurrency . . . 164
6.53 Content Filtering . . . 165
6.54 Data Availability . . . 167
6.55 Debuggability . . . 172
6.56 Discovery . . . 177
6.57 Discovery Isolation . . . 183
6.58 Durability . . . 184
6.59 Flow Control . . . 188
6.60 Fragmentation . . . 191
6.61 Instance Lifecycle . . . 193
6.62 Sending Large Data . . . 197
6.63 Liveliness . . . 199
6.64 Logging . . . 201
6.65 Middleware Memory Management . . . 204
6.66 QoS Profile Management . . . 220
6.67 Reliability QoS . . . 222
6.68 Sample Memory Management QoS . . . 230
6.69 Sample Ordering . . . 234
6.70 Security . . . 235
6.71 Throughput . . . 236
6.72 Transports . . . 237
6.73 Topic Queries . . . 239
6.74 Type System . . . 240
6.75 QoS Mutability . . . 242
6.75.1 Detailed Description . . . 242
6.76 Immutable QoS . . . 243
6.77 Mutable QoS . . . 280
6.78 Other QoS Attributes . . . 286
6.78.1 Detailed Description . . . 287
6.79 Applied Per-Instance . . . 287
6.80 Discovery Metadata . . . 288
6.81 Matching . . . 293
6.82 Deprecated QoS . . . 296

6.83 System Properties . . . 301
6.83.1 System Properties List . . . 301
6.83.2 System Resource Consideration . . . 302
6.84 Configuring QoS Profiles with XML . . . 302
6.84.1 Detailed Description . . . 302
6.84.2 Loading QoS Profiles from XML Resources . . . 302
6.84.3 URL . . . 304
6.84.3.1 URL groups . . . 304
6.84.3.2 NDDS_QOS_PROFILES environment variable . . . 304
6.84.3.3 Built-In QoS Profiles . . . 304
6.85 Publication Example . . . 304
6.85.1 A typical publication example . . . 305
6.86 Subscription Example . . . 305
6.86.1 A typical subscription example . . . 306
6.87 Participant Use Cases . . . 306
6.87.1 Turning off auto-enable of newly created participant(s) . . . 306
6.87.2 Setting up a DomainParticipant . . . 307
6.87.3 Looking up DomainParticipants . . . 307
6.87.4 Tearing down a Participant . . . 308
6.87.5 Finalizing the factory . . . 308
6.88 Topic Use Cases . . . 308
6.88.1 Registering a user data type . . . 308
6.88.2 Setting up a Topic . . . 308
6.88.3 Discovering Topics . . . 309
6.88.4 Tearing down a topic . . . 310
6.89 Publisher Use Cases . . . 310
6.89.1 Setting up a publisher . . . 310
6.89.2 Looking up Publishers . . . 310
6.89.3 Tearing down a publisher . . . 311
6.90 DataWriter Use Cases . . . 311
6.90.1 Setting up a DataWriter . . . 311
6.90.2 Managing instances . . . 312
6.90.3 Sending data . . . 312
6.90.4 DataWriter Listeners . . . 313
6.90.5 Looking up DataWriters . . . 314
6.90.6 Getting Matched Subscriptions . . . 315
6.90.7 Tearing down a DataWriter . . . 315
6.91 Subscriber Use Cases . . . 315
6.91.1 Setting up a subscriber . . . 315

6.91.2 Looking up Subscribers . . . 316
6.91.3 Tearing down a subscriber . . . 316
6.92 DataReader Use Cases . . . 316
6.92.1 Setting up a DataReader . . . 316
6.92.2 Managing instances . . . 317
6.92.3 Set up a DataReader to access received data . . . 317
6.92.4 Accessing Received Data . . . 318
6.92.4.1 Reading data samples . . . 318
6.92.4.2 Selecting what samples to read . . . 319
6.92.4.3 Reading samples using coherent access . . . 320
6.92.5 DataReader Listeners . . . 320
6.92.6 Looking up DataReaders . . . 322
6.92.7 Getting Matched Publications . . . 323
6.92.8 Accessing the Built-in Subscriber and DataReaders . . . 323
6.92.9 Using untyped DataReaders . . . 324
6.92.10 Tearing down a DataReader . . . 324
6.93 Entity Use Cases . . . 325
6.93.1 Changing the QoS for an entity . . . 325
6.93.2 Changing the listener and enabling/disabling statuses associated with it . . . 325
6.93.3 Enabling/Disabling statuses associated with a status condition . . . 327
6.93.4 Ignoring Entities . . . 328
6.93.5 Tearing Down An Entity . . . 328
6.94 Waitset Use Cases . . . 330
6.94.1 Setting up a WaitSet . . . 330
6.94.2 Waiting for Condition(s) to trigger . . . 330
6.95 Filter Use Cases . . . 331
6.95.1 Introduction . . . 331
6.95.2 Code Examples . . . 332
6.95.3 Filtering with ContentFilteredTopic . . . 332
6.95.4 Filtering with Query Conditions . . . 332
6.95.5 Filtering Performance . . . 333
6.96 Creating Custom Content Filters . . . 333
6.96.1 Introduction
 . . . 333
6.96.2 The Custom Content Filter API . . . 334
6.96.2.1 The compile function . . . 334
6.96.2.2 The evaluate function . . . 335
6.96.2.3 The finalize function . . . 335
6.96.2.4 The writer_attach function . . . 335
6.96.2.5 The writer_compile function . . . 335

6.96.2.6 The writer_evaluate function . . . 335
6.96.2.7 The writer_detach function . . . 335
6.96.2.8 The writer_return_loan function . . . 335
6.96.2.9 The writer_finalize function . . . 336
6.96.3 Example Custom Writer Content Filter . . . 336
6.96.3.1 Writing the Compile Function . . . 337
6.96.3.2 Writing the Evaluate Function . . . 337
6.96.3.3 Writing the Finalize Function . . . 338
6.96.3.4 Writing the Writer Attach Function . . . 338
6.96.3.5 Writing the Writer Compile Function . . . 338
6.96.3.6 Writing the Writer Evaluate Function . . . 339
6.96.3.7 Writing the Writer Detach Function . . . 339
6.96.3.8 Writing the Writer Return Loan Function . . . 339
6.96.3.9 Writing the Writer Finalize Function . . . 339
6.96.3.10 Creating a CustomFilter . . . 340
6.96.3.11 Registering the Filter . . . 340
6.96.3.12 Unregistering the Filter . . . 340
6.96.3.13 Using a CustomFilter . . . 340
6.96.3.14 Looking up the Filter . . . 341
6.97 XML Application Creation . . . 342
6.97.1 Introduction
 . . . 342
6.97.2 Setting up this Example . . . 343
6.97.2.1 Using XML Application Creation . . . 343
6.98 Request-Reply Examples . . . 344
6.98.1 Request-Reply Examples . . . 344
6.98.2 Requester Creation . . . 345
6.98.3 Creating a Requester with optional parameters . . . 345
6.98.4 Basic Requester example . . . 346
6.98.5 Correlating requests and replies . . . 346
6.98.6 Creating a Replier . . . 347
6.98.7 Basic Replier example . . . 348
6.98.8 SimpleReplier example . . . 348
6.98.9 Configuring Request-Reply QoS profiles . . . 348
6.99 Documentation Roadmap . . . 349
6.100 Conventions . . . 350
6.100.1 Type system . . . 350
6.100.1.1 Value types . . . 350
6.100.1.2 Reference types . . . 351
6.100.1.3 Move-only types . . . 353

6.100.2 C++11 Support . . . 353
6.100.3 Exceptions in the API . . . 353
6.100.4 Extensions to the standard API . . . 354
6.100.5 Experimental . . . 355
6.100.6 Method Parameters . . . 355
6.101 Namespaces and headers . . . 355
6.101.1 Header Files . . . 355
6.101.2 Namespaces . . . 356
6.102 RTI Connext DDS API Reference . . . 357
6.102.1 Detailed Description . . . 358
6.102.2 Overview . . . 359
6.102.3 Conceptual Model . . . 359
6.102.4 Modules . . . 361
6.103 Additional RTI Connext Communication Patterns . . . 361
6.103.1 Detailed Description . . . 362
6.104 Programming How-To's . . . 362
6.104.1 Detailed Description . . . 363
6.105 Interface . . . 363
6.105.1 Detailed Description . . . 363
6.105.2 Enumeration Type Documentation . . . 364
6.105.2.1 NDDS_Transport_Interface_Status_t . . . 364
6.106 Transport Plugins Configuration . . . 365
6.106.1 Detailed Description . . . 366
6.106.2 Macro Definition Documentation . . . 367
6.106.2.1 NDDS_TRANSPORT_PORT_INVALID . . . 367
6.106.2.2 NDDS_TRANSPORT_UUID_SIZE . . . 367
6.106.2.3 NDDS_TRANSPORT_LENGTH_UNLIMITED . . . 367
6.106.2.4 NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN . . . 367
6.106.2.5 NDDS_TRANSPORT_UUID_UNKNOWN . . . 367
6.106.2.6 NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED . . . 368
6.106.2.7 NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC 368
6.106.2.8 NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT . . . 368
6.106.2.9 NDDS_TRANSPORT_CLASSID_INVALID . . . 368
6.106.2.10 NDDS_TRANSPORT_CLASSID_UDPv4 . . . 369
6.106.2.11 NDDS_TRANSPORT_CLASSID_SHMEM . . . 369
6.106.2.12 NDDS_TRANSPORT_CLASSID_SHMEM_510 . . . 369
6.106.2.13 NDDS_TRANSPORT_CLASSID_UDPv6 . . . 369
6.106.2.14 NDDS_TRANSPORT_CLASSID_UDPv6_510 . . . 369
6.106.2.15 NDDS_TRANSPORT_CLASSID_TCPV4_LAN . . . 369

6.106.2.16 NDDS_TRANSPORT_CLASSID_TCPV4_WAN . . . 370
6.106.2.17 NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN . . . 370
6.106.2.18 NDDS_TRANSPORT_CLASSID_TLSV4_LAN . . . 370
6.106.2.19 NDDS_TRANSPORT_CLASSID_TLSV4_WAN . . . 370
6.106.2.20 NDDS_TRANSPORT_CLASSID_UDPv4_WAN . . . 370
6.106.2.21 NDDS_TRANSPORT_CLASSID_RESERVED_RANGE . . . 370
6.106.2.22 NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED . . . 371
6.106.2.23 NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN . . . 371
6.106.3 Typedef Documentation . . . 371
6.106.3.1 NDDS_Transport_Port_t . . . 371
6.106.3.2 NDDS_Transport_ClassId_t . . . 372
6.107 Transport Address . . . 372
6.107.1 Detailed Description . . . 373
6.107.2 Macro Definition Documentation . . . 373
6.107.2.1 NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER . . . 374
6.107.2.2 NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE . . . 374
6.107.3 Function Documentation . . . 374
6.107.3.1 NDDS_Transport_Address_to_string() . . . 374
6.107.3.2 NDDS_Transport_Address_to_string_with_protocol_family_format() . . . 375
6.107.3.3 NDDS_Transport_Address_from_string() . . . 376
6.107.3.4 NDDS_Transport_Address_print() . . . 376
6.107.3.5 NDDS_Transport_Address_is_ipv4() . . . 376
6.107.3.6 NDDS_Transport_Address_is_multicast() . . . 377
6.107.4 Variable Documentation . . . 377
6.107.4.1 NDDS_TRANSPORT_ADDRESS_INVALID . . . 377
6.108 UDP Transport Plugin definitions . . . 378
6.108.1 Detailed Description . . . 378
6.108.2 Macro Definition Documentation . . . 378
6.108.2.1 NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT . . . 379
6.108.2.2 NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT . . . 379
6.108.2.3 NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX . . . 379
6.108.2.4 NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT . . . 379
6.108.2.5 NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT . . . 380
6.108.2.6 NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT . . . 380
6.108.2.7 NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT . . . 380
6.108.3 Typedef Documentation . . . 380
6.108.3.1 NDDS_Transport_UDP_Port . . . 380
6.109 Shared Memory Transport . . . 380
6.109.1 Detailed Description . . . 381

6.109.2 Compatibility of Sender and Receiver Transports . . . 381
6.109.3 Crashing and Restarting Programs . . . 382
6.109.4 Shared Resource Keys . . . 382
6.109.5 Creating and Registering Shared Memory Transport Plugin . . . 382
6.109.6 Shared Memory Transport Property Names in Property QoS Policy of Domain Participant . . . 383
6.109.7 Macro Definition Documentation . . . 384
6.109.7.1 NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT . . . 384
6.109.7.2 NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT . . . 384
6.109.7.3 NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT 384
6.109.7.4 NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT . . . 384
6.109.7.5 NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT . . 385
6.109.7.6 NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT . . . 385
6.109.7.7 NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX . . . 385
6.109.7.8 NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT . . . 385
6.109.8 Function Documentation . . . 385
6.109.8.1 NDDS_Transport_Shmem_new() . . . 385
6.110 UDPv4 Transport . . . 386
6.110.1 Detailed Description . . . 387
6.110.2 UDPv4 Transport Property Names in Property QoS Policy of Domain Participant . . . 388
6.110.3 Macro Definition Documentation . . . 390
6.110.3.1 NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT . . . 390
6.110.3.2 NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT . . . 390
6.110.3.3 NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT . . . 390
6.110.3.4 NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT . 391
6.110.3.5 NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX . . . 391
6.110.3.6 NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT . . . 391
6.110.3.7 NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT . . . 391
6.110.3.8 NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT . . . 392
6.110.3.9 NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX . . . 392
6.110.3.10 NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT . . . 392
6.110.3.11 NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT . . . 392
6.110.3.12 NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER . . . 392
6.110.3.13 NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS . . . 393
6.110.3.14 NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT . . . 393
6.110.3.15 NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT . . . 393
6.110.3.16 NDDS_Transport_UDPv4_string_to_address_cEA . . . 393
6.110.4 Function Documentation . . . 393
6.110.4.1 NDDS_Transport_UDPv4_new() . . . 394
6.111 Real-Time WAN Transport . . . 394

6.111.1 Detailed Description . . . 395
6.111.2 Real-Time WAN Transport Property . . . 396
6.111.3 Macro Definition Documentation . . . 398
6.111.3.1 NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT . . . 398
6.111.3.2 NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT . . . 398
6.111.4 Function Documentation . . . 398
6.111.4.1 NDDS_Transport_UDPv4_WAN_new() . . . 399
6.112 UDPv6 Transport . . . 399
6.112.1 Detailed Description . . . 401
6.112.2 UDPv6 Transport Property Names in Property QoS Policy of Domain Participant . . . 401
6.112.3 Macro Definition Documentation . . . 403
6.112.3.1 NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT . . . 403
6.112.3.2 NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT . . . 404
6.112.3.3 NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT . 404
6.112.3.4 NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX . . . 404
6.112.3.5 NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT . . . 404
6.112.3.6 NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT . . . 404
6.112.3.7 NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT . . . 405
6.112.3.8 NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX . . . 405
6.112.3.9 NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT . . . 405
6.112.3.10 NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT . . . 405
6.112.3.11 NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER . . . 405
6.112.3.12 NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS . . . 405
6.112.3.13 NDDS_Transport_UDPv6_string_to_address_cEA . . . 405
6.112.4 Function Documentation . . . 406
6.112.4.1 NDDS_Transport_UDPv6_new() . . . 406
6.113 Client-side API . . . 407
6.113.1 Detailed Description . . . 407
6.113.2 Typedef Documentation . . . 407
6.113.2.1 RobotControlClientEndpoint . . . 407
6.114 Server-side API . . . 408
6.114.1 Detailed Description . . . 408
6.114.2 Typedef Documentation . . . 408
6.114.2.1 RobotControlService . . . 408
6.114.2.2 ServiceParams . . . 408
6.115 FlatData Builders . . . 409
6.115.1 Detailed Description . . . 409
6.115.2 Builder Error Management . . . 410
6.115.3 Function Documentation . . . 411

6.115.3.1 build_data() . . . 411
6.115.3.2 discard_builder() . . . 411
6.116 FlatData Samples . . . 412
6.116.1 Detailed Description . . . 412
6.116.2 Typedef Documentation . . . 413
6.116.2.1 MyFlatFinal . . . 413
6.116.2.2 MyFlatMutable . . . 413
6.116.2.3 MyFlatUnion . . . 414
6.117 FlatData Offsets . . . 414
6.117.1 Detailed Description . . . 415
6.117.2 Offset Error Management . . . 416
6.117.3 Function Documentation . . . 417
6.117.3.1 plain_cast() [1/2] . . . 417
6.117.3.2 plain_cast() [2/2] . . . 419
6.118 FlatData Topic-Types . . . 419
6.118.1 Detailed Description . . . 419
6.118.2 Publishing FlatData . . . 420
6.118.3 Subscribing to FlatData . . . 421
6.119 RPC Tutorial . . . 421
6.119.1 Defining a service interface in IDL . . . 422
6.119.2 Writing the Service application . . . 422
6.119.3 Writing the Client application . . . 424
6.119.4 Advanced interface definition . . . 424
6.120 Conditions and WaitSets . . . 426
6.120.1 Detailed Description . . . 426
6.121 Time Support . . . 427
6.121.1 Detailed Description . . . 427
6.122 Exceptions . . . 427
6.122.1 Detailed Description . . . 428
6.122.2 Standard Exceptions . . . 428
6.122.3 Catching exceptions . . . 428
6.123 QoS Policy Traits . . . 428
6.123.1 Detailed Description . . . 429
6.124 Safe Enumeration . . . 429
6.124.1 Detailed Description . . . 429
6.125 Status Kinds . . . 429
6.125.1 Detailed Description . . . 430
6.125.2 Changes in Status . . . 431
6.125.2.1 Changes in plain communication status . . . 432

6.125.2.2 Changes in read communication status . . . 432
6.126 Supporting Types and Constants . . . 433
6.126.1 Detailed Description . . . 435
6.126.2 Typedef Documentation . . . 435
6.126.2.1 shared_ptr . . . 435
6.126.2.2 string . . . 435
6.126.2.3 wstring . . . 435
6.126.2.4 ByteSeq . . . 436
6.126.2.5 StringSeq . . . 436
6.126.2.6 null_type . . . 436
6.126.2.7 optional . . . 436
6.126.2.8 sequence . . . 436
6.126.2.9 bounded_sequence . . . 437
6.126.2.10 string_view . . . 437
6.126.2.11 wstring_view . . . 437
6.126.3 Function Documentation . . . 437
6.126.3.1 unregister_thread() . . . 437
6.126.4 Variable Documentation . . . 438
6.126.4.1 LENGTH_UNLIMITED . . . 438
6.126.4.2 null . . . 438
6.126.4.3 length_auto . . . 439
6.126.4.4 qos_print_all . . . 439
6.127 DynamicType and DynamicData . . . 439
6.127.1 Detailed Description . . . 440
6.127.2 Typedef Documentation . . . 440
6.127.2.1 TypeKind . . . 441
6.128 Participant Built-in Topics . . . 441
6.128.1 Detailed Description . . . 441
6.128.2 Function Documentation . . . 441
6.128.2.1 participant_topic_name() . . . 441
6.129 Topic Built-in Topics . . . 442
6.129.1 Detailed Description . . . 442
6.129.2 Function Documentation . . . 442
6.129.2.1 topic_topic_name() . . . 442
6.130 Publication Built-in Topics . . . 442
6.130.1 Detailed Description . . . 443
6.130.2 Function Documentation . . . 443
6.130.2.1 publication_topic_name() . . . 443
6.131 Subscription Built-in Topics . . . 443

6.131.1 Detailed Description . . . 443
6.131.2 Function Documentation . . . 444
6.131.2.1 subscription_topic_name() . . . 444
6.132 Topic traits and data-type support . . . 444
6.132.1 Detailed Description . . . 445
6.132.2 Typedef Documentation . . . 445
6.132.2.1 PrintFormatKind . . . 445
6.132.3 Function Documentation . . . 445
6.132.3.1 to_string() [1/2] . . . 446
6.132.3.2 to_string() [2/2] . . . 447
6.132.3.3 from_string() . . . 447
6.133 Activity Context . . . 449
6.133.1 Detailed Description . . . 450
6.133.2 Function Documentation . . . 450
6.133.2.1 set_attribute_mask() . . . 451
6.134 Compliance Configuration . . . 451
6.134.1 Detailed Description . . . 451
6.134.2 Function Documentation . . . 451
6.134.2.1 get_xtypes_mask() . . . 452
6.134.2.2 set_xtypes_mask() . . . 452
6.134.2.3 load_compliance_masks() . . . 453
6.134.2.4 get_rpc_mask() . . . 454
6.134.2.5 set_rpc_mask() . . . 454
6.135 Logging . . . 455
6.135.1 Detailed Description . . . 456
6.135.2 Typedef Documentation . . . 456
6.135.2.1 LogLevel . . . 456
6.135.2.2 Verbosity . . . 457
6.135.2.3 LogCategory . . . 457
6.135.2.4 PrintFormat . . . 457
6.135.3 Enumeration Type Documentation . . . 457
6.135.3.1 SyslogLevel . . . 457
6.135.3.2 SyslogVerbosity . . . 458
6.135.3.3 LogFacility . . . 459
6.136 Version . . . 459
6.136.1 Detailed Description . . . 460
6.136.2 Function Documentation . . . 460
6.136.2.1 request_reply_api_version() . . . 460
6.136.2.2 request_reply_build_number() . . . 460

6.136.2.3 core_version() . . . 461
6.136.2.4 core_build_number() . . . 461
6.136.2.5 c_api_version() . . . 461
6.136.2.6 c_build_number() . . . 461
6.136.2.7 cpp_api_version() . . . 461
6.136.2.8 cpp_build_number() . . . 461
6.136.2.9 product_version() . . . 462
6.137 Builtin Qos Profiles . . . 462
6.137.1 Detailed Description . . . 467
6.137.2 Function Documentation . . . 469
6.137.2.1 library_name() [1/3] . . . 469
6.137.2.2 baseline() . . . 469
6.137.2.3 baseline_5_0_0() . . . 470
6.137.2.4 baseline_5_1_0() . . . 470
6.137.2.5 baseline_5_2_0() . . . 470
6.137.2.6 baseline_5_3_0() . . . 470
6.137.2.7 baseline_6_0_0() . . . 470
6.137.2.8 baseline_6_1_0() . . . 470
6.137.2.9 baseline_7_0_0() . . . 471
6.137.2.10 baseline_7_1_0() . . . 471
6.137.2.11 generic_common() . . . 471
6.137.2.12 generic_monitoring_common() . . . 471
6.137.2.13 generic_connext_micro_compatibility() . . . 472
6.137.2.14 generic_connext_micro_compatibility_2_4_9() . . . 472
6.137.2.15 generic_connext_micro_compatibility_2_4_3() . . . 472
6.137.2.16 generic_other_dds_vendor_compatibility() . . . 472
6.137.2.17 generic_510_transport_compatibility() . . . 473
6.137.2.18 generic_security() . . . 473
6.137.2.19 generic_strict_reliable() [1/2] . . . 473
6.137.2.20 generic_keep_last_reliable() [1/2] . . . 474
6.137.2.21 generic_best_effort() [1/2] . . . 474
6.137.2.22 generic_strict_reliable_high_throughput() [1/2] . . . 474
6.137.2.23 generic_strict_reliable_low_latency() [1/2] . . . 474
6.137.2.24 generic_participant_large_data() [1/2] . . . 475
6.137.2.25 generic_participant_large_data_monitoring() [1/2] . . . 475
6.137.2.26 generic_strict_reliable_large_data() [1/2] . . . 476
6.137.2.27 generic_keep_last_reliable_large_data() [1/2] . . . 476
6.137.2.28 generic_strict_reliable_large_data_fast_flow() [1/2] . . . 476
6.137.2.29 generic_strict_reliable_large_data_medium_flow() [1/2] . . . 476

6.137.2.30 generic_strict_reliable_large_data_slow_flow() [1/2] . . . 477
6.137.2.31 generic_keep_last_reliable_large_data_fast_flow() [1/2] . . . 477
6.137.2.32 generic_keep_last_reliable_large_data_medium_flow() [1/2] . . . 477
6.137.2.33 generic_keep_last_reliable_large_data_slow_flow() [1/2] . . . 477
6.137.2.34 generic_keep_last_reliable_transient_local() [1/2] . . . 478
6.137.2.35 generic_keep_last_reliable_transient() [1/2] . . . 478
6.137.2.36 generic_keep_last_reliable_persistent() [1/2] . . . 478
6.137.2.37 generic_auto_tuning() [1/2] . . . 478
6.137.2.38 generic_minimal_memory_footprint() [1/2] . . . 479
6.137.2.39 generic_monitoring2() . . . 479
6.137.2.40 generic_monitoring2_wan() . . . 480
6.137.2.41 pattern_rpc() . . . 480
6.137.2.42 pattern_periodic_data() [1/2] . . . 480
6.137.2.43 pattern_streaming() [1/2] . . . 481
6.137.2.44 pattern_reliable_streaming() [1/2] . . . 481
6.137.2.45 pattern_event() [1/2] . . . 481
6.137.2.46 pattern_alarm_event() [1/2] . . . 482
6.137.2.47 pattern_status() [1/2] . . . 482
6.137.2.48 pattern_alarm_status() [1/2] . . . 482
6.137.2.49 pattern_last_value_cache() [1/2] . . . 482
6.137.2.50 library_name() [2/3] . . . 483
6.137.2.51 generic_strict_reliable() [2/2] . . . 483
6.137.2.52 generic_keep_last_reliable() [2/2] . . . 483
6.137.2.53 generic_best_effort() [2/2] . . . 483
6.137.2.54 generic_strict_reliable_high_throughput() [2/2] . . . 484
6.137.2.55 generic_strict_reliable_low_latency() [2/2] . . . 484
6.137.2.56 generic_participant_large_data() [2/2] . . . 484
6.137.2.57 generic_participant_large_data_monitoring() [2/2] . . . 485
6.137.2.58 generic_strict_reliable_large_data() [2/2] . . . 485
6.137.2.59 generic_keep_last_reliable_large_data() [2/2] . . . 486
6.137.2.60 generic_strict_reliable_large_data_fast_flow() [2/2] . . . 486
6.137.2.61 generic_strict_reliable_large_data_medium_flow() [2/2] . . . 486
6.137.2.62 generic_strict_reliable_large_data_slow_flow() [2/2] . . . 486
6.137.2.63 generic_keep_last_reliable_large_data_fast_flow() [2/2] . . . 487
6.137.2.64 generic_keep_last_reliable_large_data_medium_flow() [2/2] . . . 487
6.137.2.65 generic_keep_last_reliable_large_data_slow_flow() [2/2] . . . 487
6.137.2.66 generic_keep_last_reliable_transient_local() [2/2] . . . 487
6.137.2.67 generic_keep_last_reliable_transient() [2/2] . . . 488
6.137.2.68 generic_keep_last_reliable_persistent() [2/2] . . . 488

6.137.2.69 generic_auto_tuning() [2/2] . . . 488
6.137.2.70 generic_minimal_memory_footprint() [2/2] . . . 488
6.137.2.71 pattern_periodic_data() [2/2] . . . 489
6.137.2.72 pattern_streaming() [2/2] . . . 489
6.137.2.73 pattern_reliable_streaming() [2/2] . . . 489
6.137.2.74 pattern_event() [2/2] . . . 490
6.137.2.75 pattern_alarm_event() [2/2] . . . 490
6.137.2.76 pattern_status() [2/2] . . . 490
6.137.2.77 pattern_alarm_status() [2/2] . . . 491
6.137.2.78 pattern_last_value_cache() [2/2] . . . 491
6.137.2.79 library_name() [3/3] . . . 491
6.137.2.80 snippet_optimization_reliability_protocol_common() . . . 491
6.137.2.81 snippet_optimization_reliability_protocol_keep_all() . . . 492
6.137.2.82 snippet_optimization_reliability_protocol_keep_last() . . . 492
6.137.2.83 snippet_optimization_reliability_protocol_high_rate() . . . 493
6.137.2.84 snippet_optimization_reliability_protocol_low_latency() . . . 493
6.137.2.85 snippet_optimization_reliability_protocol_large_data() . . . 494
6.137.2.86 snippet_optimization_reliability_protocol_dynamicmemalloc() . . . 494
6.137.2.87 snippet_optimization_discovery_common() . . . 495
6.137.2.88 snippet_optimization_discovery_participant_compact() . . . 495
6.137.2.89 snippet_optimization_discovery_endpoint_fast() . . . 495
6.137.2.90 snippet_optimization_transport_large_buffers() . . . 496
6.137.2.91 snippet_qos_policy_reliability_reliable() . . . 496
6.137.2.92 snippet_qos_policy_reliability_best_effort() . . . 497
6.137.2.93 snippet_qos_policy_history_keep_last_1() . . . 497
6.137.2.94 snippet_qos_policy_history_keep_all() . . . 497
6.137.2.95 snippet_qos_policy_publish_mode_asynchronous() . . . 498
6.137.2.96 snippet_qos_policy_durability_transient_local() . . . 498
6.137.2.97 snippet_qos_policy_durability_transient() . . . 498
6.137.2.98 snippet_qos_policy_durability_persistent() . . . 499
6.137.2.99 snippet_qos_policy_batching_enable() . . . 499
6.137.2.100 snippet_qos_policy_flow_controller_838mbps() . . . 499
6.137.2.101 snippet_qos_policy_flow_controller_209mbps() . . . 500
6.137.2.102 snippet_qos_policy_flow_controller_52mbps() . . . 500
6.137.2.103 snippet_feature_auto_tuning_enable() . . . 501
6.137.2.104 snippet_feature_monitoring_enable() . . . 501
6.137.2.105 snippet_feature_monitoring2_enable() . . . 502
6.137.2.106 snippet_feature_security_enable() . . . 502
6.137.2.107 snippet_feature_topic_query_enable() . . . 502

6.137.2.108 snippet_feature_type_support_v1_enable() . . . 503
6.137.2.109 snippet_feature_type_support_v1_disable() . . . 503
6.137.2.110 snippet_feature_type_support_v2_enable() . . . 503
6.137.2.111 snippet_feature_type_support_v2_disable() . . . 504
6.137.2.112 snippet_transport_tcp_lan_client() . . . 504
6.137.2.113 snippet_transport_tcp_wan_symmetric_client() . . . 504
6.137.2.114 snippet_transport_tcp_wan_asymmetric_server() . . . 505
6.137.2.115 snippet_transport_tcp_wan_asymmetric_client() . . . 505
6.137.2.116 snippet_transport_udp_avoid_ip_fragmentation() . . . 505
6.137.2.117 snippet_transport_udp_wan() . . . 506
6.137.2.118 snippet_compatibility_connext_micro_version_2_4_3() . . . 506
6.137.2.119 snippet_compatibility_other_dds_vendors_enable() . . . 506
6.137.2.120 snippet_compatibility_5_1_0_transport_enable() . . . 507
6.138 AsyncWaitSet . . . 507
6.138.1 Detailed Description . . . 507
6.138.2 Function Documentation . . . 508
6.138.2.1 Ignore() . . . 508
6.139 QoS . . . 508
6.139.1 Detailed Description . . . 508
6.140 All QoS Policies . . . 508
6.140.1 Detailed Description . . . 512
6.140.2 Specifying QoS on entities . . . 513
6.140.3 QoS compatibility . . . 513
6.140.4 Typedef Documentation . . . 514
6.140.4.1 QosPolicyCountSeq . . . 514
6.140.4.2 QosPolicyId . . . 514
6.141 QoS by Category . . . 514
6.141.1 Detailed Description . . . 515
6.142 ASYNCHRONOUS_PUBLISHER . . . 515
6.142.1 Detailed Description . . . 516
6.143 AVAILABILITY . . . 516
6.143.1 Detailed Description . . . 516
6.143.2 Typedef Documentation . . . 516
6.143.2.1 EndpointGroupSeq . . . 516
6.144 BATCH . . . 517
6.144.1 Detailed Description . . . 517
6.145 DATABASE . . . 517
6.145.1 Detailed Description . . . 517
6.146 DATA_READER_PROTOCOL . . . 517

6.146.1 Detailed Description . . . 518
6.147 DATA_READER_RESOURCE_LIMITS . . . 518
6.147.1 Detailed Description . . . 518
6.147.2 Typedef Documentation . . . 518
6.147.2.1 DataReaderInstanceRemovalKind . . . 519
6.147.3 Function Documentation . . . 519
6.147.3.1 auto_max_total_instances() . . . 519
6.148 DATA_REPRESENTATION . . . 519
6.148.1 Detailed Description . . . 520
6.148.2 Typedef Documentation . . . 520
6.148.2.1 DataRepresentationId . . . 520
6.148.2.2 DataRepresentationIdSeq . . . 520
6.148.3 Function Documentation . . . 520
6.148.3.1 xcdr() . . . 520
6.148.3.2 xml() . . . 521
6.148.3.3 xcdr2() . . . 521
6.148.3.4 auto_id() . . . 521
6.149 DATA_WRITER_PROTOCOL . . . 521
6.149.1 Detailed Description . . . 522
6.150 DATA_WRITER_RESOURCE_LIMITS . . . 522
6.150.1 Detailed Description . . . 522
6.150.2 Typedef Documentation . . . 522
6.150.2.1 DataWriterResourceLimitsInstanceReplacementKind . . . 522
6.151 DATA_WRITER_TRANSFER_MODE . . . 523
6.151.1 Detailed Description . . . 523
6.152 DATA_TAG . . . 523
6.152.1 Detailed Description . . . 523
6.153 DEADLINE . . . 523
6.153.1 Detailed Description . . . 524
6.154 DESTINATION_ORDER . . . 524
6.154.1 Detailed Description . . . 524
6.154.2 Typedef Documentation . . . 524
6.154.2.1 DestinationOrderKind . . . 525
6.154.2.2 DestinationOrderScopeKind . . . 525
6.155 DISCOVERY . . . 525
6.155.1 Detailed Description . . . 525
6.156 DISCOVERY_CONFIG . . . 526
6.156.1 Detailed Description . . . 526
6.156.2 Typedef Documentation . . . 526

6.156.2.1 RemoteParticipantPurgeKind . . . 527
6.156.3 Function Documentation . . . 527
6.156.3.1 all() . . . 527
6.156.3.2 none() . . . 527
6.157 DOMAIN_PARTICIPANT_RESOURCE_LIMITS . . . 528
6.157.1 Detailed Description . . . 528
6.157.2 Typedef Documentation . . . 528
6.157.2.1 IgnoredEntityReplacementKind . . . 528
6.158 DURABILITY . . . 529
6.158.1 Detailed Description . . . 530
6.158.2 Typedef Documentation . . . 530
6.158.2.1 DurabilityKind . . . 530
6.158.3 Enumeration Type Documentation . . . 530
6.158.3.1 PersistentJournalKind . . . 530
6.158.3.2 PersistentSynchronizationKind . . . 531
6.158.4 Function Documentation . . . 531
6.158.4.1 auto_writer_depth() . . . 531
6.159 DURABILITY_SERVICE . . . 531
6.159.1 Detailed Description . . . 532
6.160 ENTITY_FACTORY . . . 532
6.160.1 Detailed Description . . . 532
6.161 ENTITY_NAME . . . 532
6.161.1 Detailed Description . . . 533
6.162 EVENT . . . 533
6.162.1 Detailed Description . . . 533
6.163 HISTORY . . . 533
6.163.1 Detailed Description . . . 533
6.163.2 Typedef Documentation . . . 534
6.163.2.1 HistoryKind . . . 534
6.164 GROUP_DATA . . . 534
6.164.1 Detailed Description . . . 534
6.165 LATENCY_BUDGET . . . 534
6.165.1 Detailed Description . . . 534
6.166 LIFESPAN . . . 535
6.166.1 Detailed Description . . . 535
6.167 LIVELINESS . . . 535
6.167.1 Detailed Description . . . 535
6.167.2 Typedef Documentation . . . 535
6.167.2.1 LivelinessKind . . . 536

6.168 LOCATORFILTER . . . 536
6.168.1 Detailed Description . . . 536
6.169 LOGGING . . . 536
6.170 MONITORING . . . 536
6.170.1 Detailed Description . . . 537
6.171 MULTICHANNEL . . . 537
6.171.1 Detailed Description . . . 538
6.171.2 Typedef Documentation . . . 538
6.171.2.1 ChannelSettingsSeq . . . 538
6.172 OWNERSHIP . . . 538
6.172.1 Detailed Description . . . 538
6.172.2 Typedef Documentation . . . 539
6.172.2.1 OwnershipKind . . . 539
6.173 OWNERSHIP_STRENGTH . . . 539
6.173.1 Detailed Description . . . 539
6.174 PARTITION . . . 539
6.174.1 Detailed Description . . . 540
6.175 PRESENTATION . . . 540
6.175.1 Detailed Description . . . 540
6.175.2 Typedef Documentation . . . 540
6.175.2.1 PresentationAccessScopeKind . . . 540
6.176 PROFILE . . . 541
6.176.1 Detailed Description . . . 541
6.177 PROPERTY . . . 541
6.177.1 Detailed Description . . . 541
6.178 PUBLISH_MODE . . . 541
6.178.1 Detailed Description . . . 542
6.178.2 Typedef Documentation . . . 542
6.178.2.1 PublishModeKind . . . 542
6.178.3 Variable Documentation . . . 542
6.178.3.1 PUBLICATION_PRIORITY_UNDEFINED . . . 543
6.178.3.2 PUBLICATION_PRIORITY_AUTOMATIC . . . 543
6.179 READER_DATA_LIFECYCLE . . . 543
6.179.1 Detailed Description . . . 543
6.180 RECEIVER_POOL . . . 543
6.180.1 Detailed Description . . . 544
6.181 RELIABILITY . . . 544
6.181.1 Detailed Description . . . 544
6.181.2 Typedef Documentation . . . 545

6.181.2.1 ReliabilityKind . . . 545
6.181.2.2 AcknowledgmentKind . . . 545
6.181.3 Enumeration Type Documentation . . . 545
6.181.3.1 InstanceStateConsistencyKind . . . 545
6.182 RESOURCE_LIMITS . . . 546
6.182.1 Detailed Description . . . 546
6.183 SERVICE . . . 546
6.183.1 Detailed Description . . . 546
6.183.2 Typedef Documentation . . . 546
6.183.2.1 ServiceKind . . . 547
6.184 SYSTEM_RESOURCE_LIMITS . . . 547
6.184.1 Detailed Description . . . 547
6.185 TIME_BASED_FILTER . . . 547
6.185.1 Detailed Description . . . 547
6.186 TOPIC_DATA . . . 548
6.186.1 Detailed Description . . . 548
6.187 TOPIC_QUERY_DISPATCH . . . 548
6.187.1 Detailed Description . . . 548
6.188 TRANSPORT_BUILTIN . . . 548
6.188.1 Detailed Description . . . 549
6.189 TRANSPORT_MULTICAST . . . 549
6.189.1 Detailed Description . . . 549
6.189.2 Typedef Documentation . . . 549
6.189.2.1 TransportMulticastKind . . . 550
6.189.2.2 TransportMulticastSettingsSeq . . . 550
6.190 TRANSPORT_MULTICAST_MAPPING . . . 550
6.190.1 Detailed Description . . . 550
6.191 TRANSPORT_PRIORITY . . . 550
6.191.1 Detailed Description . . . 551
6.192 TRANSPORT_SELECTION . . . 551
6.192.1 Detailed Description . . . 551
6.193 TRANSPORT_UNICAST . . . 551
6.193.1 Detailed Description . . . 552
6.193.2 Typedef Documentation . . . 552
6.193.2.1 TransportUnicastSettingsSeq . . . 552
6.194 TYPE_CONSISTENCY_ENFORCEMENT . . . 552
6.194.1 Detailed Description . . . 553
6.194.2 Typedef Documentation . . . 553
6.194.2.1 TypeConsistencyEnforcementKind . . . 553

6.195 TYPESUPPORT . . . 553
6.195.1 Detailed Description . . . 553
6.195.2 Typedef Documentation . . . 554
6.195.2.1 CdrPaddingKind . . . 554
6.196 USER_DATA . . . 554
6.196.1 Detailed Description . . . 554
6.197 WRITER_DATA_LIFECYCLE . . . 554
6.197.1 Detailed Description . . . 554
6.198 WIRE_PROTOCOL . . . 555
6.198.1 Detailed Description . . . 555
6.198.2 Typedef Documentation . . . 555
6.198.2.1 WireProtocolAutoKind . . . 556
6.198.3 Function Documentation . . . 556
6.198.3.1 all() . . . 556
6.198.3.2 none() . . . 557
6.198.3.3 default_mask() . . . 557
6.198.3.4 Interoperable() . . . 558
6.198.3.5 BackwardsCompatible() . . . 558
6.199 NDDS_DISCOVERY_PEERS . . . 559
6.199.1 Peer Descriptor Format . . . 560
6.199.1.1 Locator Format . . . 560
6.199.1.2 Address Format . . . 561
6.199.2 NDDS_DISCOVERY_PEERS Environment Variable Format . . . 561
6.199.3 NDDS_DISCOVERY_PEERS File Format . . . 563
6.199.4 NDDS_DISCOVERY_PEERS Precedence . . . 564
6.199.5 NDDS_DISCOVERY_PEERS Default Value . . . 564
6.199.6 Builtin Transport Class Names . . . 565
6.199.7 NDDS_DISCOVERY_PEERS and Local Host Communication . . . 565
6.200 SampleProcessor . . . 566
6.200.1 Detailed Description . . . 566
6.201 ServiceRequest Built-in Topic . . . 566
6.201.1 Detailed Description . . . 567
6.201.2 Typedef Documentation . . . 567
6.201.2.1 ServiceRequestId . . . 567
6.201.3 Function Documentation . . . 568
6.201.3.1 service_request_topic_name() . . . 568
6.202 Topic-type serialization and deserialization . . . 568
6.202.1 Detailed Description . . . 568
6.202.2 Function Documentation . . . 568

6.202.2.1 from_cdr_buffer_no_alloc() . . . 569
6.202.2.2 from_cdr_buffer() . . . 569
6.202.2.3 to_cdr_buffer() [1/2] . . . 570
6.202.2.4 to_cdr_buffer() [2/2] . . . 570
6.203 Custom Content Filters . . . 571
6.203.1 Detailed Description . . . 572
6.203.2 Function Documentation . . . 572
6.203.2.1 find_content_filter() . . . 572
6.203.3 Variable Documentation . . . 572
6.203.3.1 no_compile_data . . . 573
6.204 Discovery Snapshot . . . 573
6.204.1 Detailed Description . . . 573
6.204.2 Function Documentation . . . 573
6.204.2.1 take_snapshot() [1/6] . . . 573
6.204.2.2 take_snapshot() [2/6] . . . 574
6.204.2.3 take_snapshot() [3/6] . . . 575
6.204.2.4 take_snapshot() [4/6] . . . 576
6.204.2.5 take_snapshot() [5/6] . . . 577
6.204.2.6 take_snapshot() [6/6] . . . 577
6.205 Network Capture . . . 578
6.205.1 Detailed Description . . . 579
6.205.2 Capturing . . . 580
6.205.3 Function Documentation . . . 581
6.205.3.1 enable() . . . 581
6.205.3.2 disable() . . . 582
6.205.3.3 set_default_params() . . . 582
6.205.3.4 start() [1/4] . . . 583
6.205.3.5 start() [2/4] . . . 584
6.205.3.6 start() [3/4] . . . 584
6.205.3.7 start() [4/4] . . . 585
6.205.3.8 stop() [1/2] . . . 586
6.205.3.9 stop() [2/2] . . . 587
6.205.3.10 pause() [1/2] . . . 587
6.205.3.11 pause() [2/2] . . . 588
6.205.3.12 resume() [1/2] . . . 588
6.205.3.13 resume() [2/2] . . . 589
6.206 Heap Monitoring . . . 589
6.206.1 Detailed Description . . . 590
6.206.2 Typedef Documentation . . . 590

6.206.2.1 SnapshotOutputFormat . . . 590
6.206.2.2 SnapshotContentFormat . . . 591
6.206.3 Function Documentation . . . 591
6.206.3.1 enable() [1/2] . . . 591
6.206.3.2 enable() [2/2] . . . 592
6.206.3.3 disable() . . . 592
6.206.3.4 pause() . . . 593
6.206.3.5 resume() . . . 593
6.206.3.6 take_snapshot() . . . 594
6.207 Other Utilities . . . 595
6.207.1 Detailed Description . . . 596
6.207.2 Function Documentation . . . 596
6.207.2.1 enable_automatic_tss_cleanup() . . . 596
6.207.2.2 sleep() . . . 596
6.207.2.3 spin_per_microsecond() . . . 597
6.207.2.4 spin() . . . 597
6.207.2.5 unload_activation_library() . . . 598
6.208 Monitoring Library 2.0 . . . 598
6.208.1 Detailed Description . . . 599
6.208.2 Function Documentation . . . 599
6.208.2.1 RTI_Monitoring_initialize() . . . 599
6.208.3 Variable Documentation . . . 600
6.208.3.1 RTI_MONITORING_PERIODIC_TOPIC_NAME . . . 600
6.208.3.2 RTI_MONITORING_EVENT_TOPIC_NAME . . . 600
6.208.3.3 RTI_MONITORING_LOGGING_TOPIC_NAME . . . 600
6.209 Built-in Types Examples . . . 601
6.210 Exceptions . . . 601
6.210.1 Exceptions . . . 602
6.211 Qos Use Cases . . . 602
6.211.1 Setting Qos Values . . . 603
6.211.2 Getting Qos Values . . . 603
6.212 Qos Provider Use Cases . . . 604
6.212.1 Managing Qos Profiles . . . 604
6.212.2 The Default Qos Provider . . . 605
6.213 Working with IDL types . . . 606
6.213.1 Detailed Description . . . 606
6.213.2 Example IDL types . . . 606
6.213.3 Constructors and operators . . . 607
6.213.4 Accessing the type members . . . 607

6.213.5 Enumerations . . . 608
6.213.6 Unions . . . 608
6.214 DynamicType and DynamicData Use Cases . . . 609
6.214.1 Creating a DynamicType . . . 609
6.214.1.1 Create a DynamicType by instantiating it and adding members . . . 609
6.214.1.2 Create a DynamicType from an XML description . . . 610
6.214.1.3 Get the DynamicType from its equivalent IDL-generated type . . . 610
6.214.1.4 Create a DynamicType using tuples . . . 610
6.214.2 Using DynamicData . . . 611
6.214.2.1 Publishing data using DynamicData . . . 611
6.214.2.2 Subscribing to data using DynamicData . . . 612
6.214.2.3 Manipulating data reflectively using C++ tuples . . . 612

7 Namespace Documentation **615**

7.1 dds Namespace Reference . . . 615
7.1.1 Detailed Description . . . 615
7.2 dds::all Namespace Reference . . . 615
7.2.1 Detailed Description . . . 616
7.3 dds::core Namespace Reference . . . 616
7.3.1 Detailed Description . . . 619
7.3.2 Typedef Documentation . . . 619
7.3.2.1 InstanceHandleSeq . . . 620
7.3.3 Function Documentation . . . 620
7.3.3.1 polymorphic_cast() . . . 620
7.3.3.2 operator∗() [1/2] . . . 621
7.3.3.3 operator∗() [2/2] . . . 621
7.3.3.4 operator/() . . . 621
7.3.3.5 swap() . . . 622
7.3.3.6 operator<<() [1/4] . . . 622
7.3.3.7 operator<<() [2/4] . . . 622
7.3.3.8 operator+() [1/2] . . . 623
7.3.3.9 operator+() [2/2] . . . 623
7.3.3.10 operator-() [1/2] . . . 623
7.3.3.11 operator-() [2/2] . . . 624
7.3.3.12 operator==() [1/3] . . . 624
7.3.3.13 operator"!=() [1/3] . . . 625
7.3.3.14 operator==() [2/3] . . . 625
7.3.3.15 operator==() [3/3] . . . 625
7.3.3.16 operator"!=() [2/3] . . . 626

7.3.3.17 operator"!=() [3/3] . . . 626
7.3.3.18 operator<<() [3/4] . . . 626
7.3.3.19 operator<<() [4/4] . . . 626
7.4 dds::core::cond Namespace Reference . . . 627
7.4.1 Detailed Description . . . 627
7.5 dds::core::policy Namespace Reference . . . 627
7.5.1 Detailed Description . . . 629
7.6 dds::core::status Namespace Reference . . . 630
7.6.1 Detailed Description . . . 630
7.6.2 Function Documentation . . . 631
7.6.2.1 get_status() . . . 631
7.7 dds::core::xtypes Namespace Reference . . . 631
7.7.1 Detailed Description . . . 632
7.7.2 Typedef Documentation . . . 633
7.7.2.1 ExtensibilityKind . . . 633
7.7.3 Function Documentation . . . 633
7.7.3.1 is_primitive_type() . . . 633
7.7.3.2 is_constructed_type() . . . 633
7.7.3.3 is_collection_type() . . . 634
7.7.3.4 is_aggregation_type() . . . 634
7.7.3.5 primitive_type() . . . 634
7.7.3.6 operator<<() . . . 634
7.8 dds::domain Namespace Reference . . . 634
7.8.1 Detailed Description . . . 635
7.8.2 Function Documentation . . . 635
7.8.2.1 ignore() [1/2] . . . 636
7.8.2.2 ignore() [2/2] . . . 636
7.8.2.3 discovered_participants() [1/2] . . . 637
7.8.2.4 discovered_participants() [2/2] . . . 638
7.8.2.5 discovered_participant_data() . . . 638
7.8.2.6 find() . . . 639
7.9 dds::domain::qos Namespace Reference . . . 640
7.9.1 Detailed Description . . . 641
7.9.2 Function Documentation . . . 641
7.9.2.1 to_string() [1/6] . . . 641
7.9.2.2 to_string() [2/6] . . . 642
7.9.2.3 to_string() [3/6] . . . 642
7.9.2.4 operator<<() [1/2] . . . 643
7.9.2.5 to_string() [4/6] . . . 643

7.9.2.6 to_string() [5/6] . . . 644
7.9.2.7 to_string() [6/6] . . . 645
7.9.2.8 operator<<() [2/2] . . . 645
7.10 dds::pub Namespace Reference . . . 645
7.10.1 Detailed Description . . . 647
7.10.2 Function Documentation . . . 647
7.10.2.1 get() . . . 647
7.10.2.2 ignore() [1/2] . . . 647
7.10.2.3 ignore() [2/2] . . . 648
7.10.2.4 matched_subscriptions() [1/2] . . . 648
7.10.2.5 matched_subscriptions() [2/2] . . . 649
7.10.2.6 matched_subscription_data() . . . 650
7.10.2.7 find() [1/2] . . . 651
7.10.2.8 find() [2/2] . . . 652
7.11 dds::pub::qos Namespace Reference . . . 653
7.11.1 Detailed Description . . . 654
7.11.2 Function Documentation . . . 654
7.11.2.1 to_string() [1/6] . . . 654
7.11.2.2 to_string() [2/6] . . . 655
7.11.2.3 to_string() [3/6] . . . 655
7.11.2.4 operator<<() [1/2] . . . 656
7.11.2.5 swap() . . . 656
7.11.2.6 to_string() [4/6] . . . 656
7.11.2.7 to_string() [5/6] . . . 657
7.11.2.8 to_string() [6/6] . . . 658
7.11.2.9 operator<<() [2/2] . . . 658
7.12 dds::sub Namespace Reference . . . 658
7.12.1 Detailed Description . . . 661
7.12.2 Function Documentation . . . 661
7.12.2.1 get() . . . 661
7.12.2.2 read() . . . 662
7.12.2.3 take() . . . 662
7.12.2.4 max_samples() . . . 662
7.12.2.5 content() . . . 663
7.12.2.6 condition() . . . 663
7.12.2.7 state() . . . 664
7.12.2.8 instance() . . . 665
7.12.2.9 next_instance() . . . 665
7.12.2.10 ignore() [1/2] . . . 667

7.12.2.11 ignore() [2/2] . . . 668
7.12.2.12 matched_publications() [1/2] . . . 668
7.12.2.13 matched_publications() [2/2] . . . 669
7.12.2.14 matched_publication_data() . . . 670
7.12.2.15 builtin_subscriber() . . . 671
7.12.2.16 find() [1/6] . . . 672
7.12.2.17 find() [2/6] . . . 673
7.12.2.18 find() [3/6] . . . 675
7.12.2.19 find() [4/6] . . . 675
7.12.2.20 find() [5/6] . . . 676
7.12.2.21 find() [6/6] . . . 677
7.12.2.22 move() . . . 678
7.12.2.23 begin() [1/2] . . . 679
7.12.2.24 begin() [2/2] . . . 679
7.12.2.25 end() [1/2] . . . 679
7.12.2.26 end() [2/2] . . . 680
7.12.2.27 swap() . . . 680
7.13 dds::sub::cond Namespace Reference . . . 680
7.13.1 Detailed Description . . . 680
7.14 dds::sub::qos Namespace Reference . . . 681
7.14.1 Detailed Description . . . 681
7.14.2 Function Documentation . . . 681
7.14.2.1 to_string() [1/6] . . . 682
7.14.2.2 to_string() [2/6] . . . 683
7.14.2.3 to_string() [3/6] . . . 684
7.14.2.4 operator<<() [1/2] . . . 684
7.14.2.5 to_string() [4/6] . . . 684
7.14.2.6 to_string() [5/6] . . . 685
7.14.2.7 to_string() [6/6] . . . 686
7.14.2.8 operator<<() [2/2] . . . 686
7.15 dds::sub::status Namespace Reference . . . 686
7.15.1 Detailed Description . . . 687
7.15.2 Function Documentation . . . 687
7.15.2.1 operator<<() [1/4] . . . 687
7.15.2.2 operator<<() [2/4] . . . 688
7.15.2.3 operator<<() [3/4] . . . 688
7.15.2.4 operator<<() [4/4] . . . 688
7.16 dds::topic Namespace Reference . . . 688
7.16.1 Detailed Description . . . 690

7.16.2 Function Documentation . . . 690
7.16.2.1 get() . . . 690
7.16.2.2 discover_any_topic() [1/2] . . . 691
7.16.2.3 discover_any_topic() [2/2] . . . 691
7.16.2.4 discover_topic_data() [1/4] . . . 691
7.16.2.5 discover_topic_data() [2/4] . . . 692
7.16.2.6 discover_topic_data() [3/4] . . . 692
7.16.2.7 discover_topic_data() [4/4] . . . 693
7.16.2.8 ignore() [1/2] . . . 693
7.16.2.9 ignore() [2/2] . . . 694
7.16.2.10 find() . . . 694
7.17 dds::topic::qos Namespace Reference . . . 695
7.17.1 Detailed Description . . . 696
7.17.2 Function Documentation . . . 696
7.17.2.1 to_string() [1/3] . . . 696
7.17.2.2 to_string() [2/3] . . . 697
7.17.2.3 to_string() [3/3] . . . 697
7.17.2.4 operator<<() . . . 698
7.18 rti Namespace Reference . . . 698
7.18.1 Detailed Description . . . 699
7.19 rti::all Namespace Reference . . . 699
7.19.1 Detailed Description . . . 699
7.20 rti::config Namespace Reference . . . 699
7.20.1 Detailed Description . . . 701
7.21 rti::core Namespace Reference . . . 701
7.21.1 Detailed Description . . . 706
7.21.2 Typedef Documentation . . . 706
7.21.2.1 LocatorKind . . . 706
7.21.2.2 MonitoringMetricSelectionSeq . . . 706
7.21.2.3 ThreadSettingsCpuRotationKind . . . 706
7.21.2.4 TransportClassId . . . 707
7.21.2.5 TransportInfoSeq . . . 707
7.21.3 Function Documentation . . . 707
7.21.3.1 fill_array() [1/2] . . . 707
7.21.3.2 fill_array() [2/2] . . . 707
7.21.3.3 initialize_native_array() . . . 708
7.21.3.4 operator<<() [1/2] . . . 708
7.21.3.5 bind_listener() [1/2] . . . 708
7.21.3.6 bind_listener() [2/2] . . . 709

7.21.3.7 bind_and_manage_listener() [1/2] . . . 709
7.21.3.8 bind_and_manage_listener() [2/2] . . . 710
7.21.3.9 operator<<() [2/2] . . . 710
7.21.3.10 default_qos_provider_params() [1/2] . . . 710
7.21.3.11 default_qos_provider_params() [2/2] . . . 711
7.22 rti::core::builtin_profiles Namespace Reference . . . 711
7.22.1 Detailed Description . . . 711
7.23 rti::core::builtin_profiles::qos_lib Namespace Reference . . . 712
7.23.1 Detailed Description . . . 714
7.24 rti::core::builtin_profiles::qos_lib_exp Namespace Reference . . . 714
7.24.1 Detailed Description . . . 716
7.25 rti::core::builtin_profiles::qos_snippet_lib Namespace Reference . . . 716
7.25.1 Detailed Description . . . 718
7.26 rti::core::xtypes Namespace Reference . . . 718
7.26.1 Detailed Description . . . 720
7.26.2 Enumeration Type Documentation . . . 720
7.26.2.1 DynamicTypePrintKind . . . 720
7.26.3 Function Documentation . . . 720
7.26.3.1 convert() [1/3] . . . 720
7.26.3.2 resolve_alias() . . . 721
7.26.3.3 to_cdr_buffer() . . . 721
7.26.3.4 from_cdr_buffer() . . . 722
7.26.3.5 convert() [2/3] . . . 722
7.26.3.6 convert() [3/3] . . . 723
7.26.3.7 can_convert() . . . 723
7.26.3.8 get_tuple() . . . 724
7.26.3.9 set_tuple() . . . 725
7.26.3.10 create_type_from_tuple() . . . 725
7.26.3.11 print_idl() . . . 726
7.26.3.12 to_string() [1/2] . . . 726
7.26.3.13 to_string() [2/2] . . . 726
7.27 rti::domain Namespace Reference . . . 727
7.27.1 Detailed Description . . . 728
7.27.2 Function Documentation . . . 728
7.27.2.1 banish_ignored_participants() . . . 728
7.27.2.2 discovered_participant_subject_name() . . . 729
7.27.2.3 discovered_participants_from_subject_name() . . . 730
7.27.2.4 find_participant_by_name() . . . 731
7.27.2.5 find_participants() [1/2] . . . 731

7.27.2.6 find_participants() [2/2] . . . 732
7.27.2.7 find_type() . . . 732
7.27.2.8 register_type() [1/2] . . . 733
7.27.2.9 register_type() [2/2] . . . 734
7.28 rti::flat Namespace Reference . . . 735
7.28.1 Detailed Description . . . 736
7.29 rti::pub Namespace Reference . . . 736
7.29.1 Detailed Description . . . 738
7.29.2 Function Documentation . . . 738
7.29.2.1 implicit_publisher() . . . 739
7.29.2.2 matched_subscription_participant_data() . . . 739
7.29.2.3 matched_subscriptions_locators() . . . 741
7.29.2.4 is_matched_subscription_active() . . . 742
7.29.2.5 matched_subscription_data() . . . 742
7.29.2.6 find_publishers() [1/2] . . . 743
7.29.2.7 find_publishers() [2/2] . . . 744
7.29.2.8 find_publisher() . . . 744
7.29.2.9 find_datawriters() [1/2] . . . 746
7.29.2.10 find_datawriters() [2/2] . . . 747
7.29.2.11 find_datawriter_by_topic_name() . . . 747
7.29.2.12 find_datawriter_by_name() [1/2] . . . 748
7.29.2.13 find_datawriter_by_name() [2/2] . . . 749
7.29.2.14 find_flow_controller() . . . 750
7.30 rti::sub Namespace Reference . . . 751
7.30.1 Detailed Description . . . 754
7.30.2 Function Documentation . . . 754
7.30.2.1 implicit_subscriber() . . . 754
7.30.2.2 matched_publication_participant_data() . . . 755
7.30.2.3 is_matched_publication_alive() . . . 756
7.30.2.4 matched_publication_data() . . . 756
7.30.2.5 find_subscribers() [1/2] . . . 757
7.30.2.6 find_subscribers() [2/2] . . . 758
7.30.2.7 find_subscriber() . . . 758
7.30.2.8 find_datareaders() [1/2] . . . 759
7.30.2.9 find_datareaders() [2/2] . . . 760
7.30.2.10 find_datareader_by_topic_name() . . . 760
7.30.2.11 find_datareader_by_name() [1/2] . . . 762
7.30.2.12 find_datareader_by_topic_description() . . . 763
7.30.2.13 find_datareader_by_name() [2/2] . . . 763

7.30.2.14 copy_to_sample() . . . 765
7.30.2.15 operator<<() . . . 765
7.30.2.16 begin() [1/2] . . . 765
7.30.2.17 begin() [2/2] . . . 766
7.30.2.18 end() [1/2] . . . 766
7.30.2.19 end() [2/2] . . . 766
7.30.2.20 swap() . . . 766
7.30.2.21 valid_data() [1/2] . . . 766
7.30.2.22 operator==() . . . 767
7.30.2.23 valid_data() [2/2] . . . 768
7.30.2.24 create_topic_query_data_from_service_request() . . . 768
7.30.2.25 find_topic_query() . . . 769
7.30.2.26 unpack() [1/3] . . . 769
7.30.2.27 unpack() [2/3] . . . 770
7.30.2.28 unpack() [3/3] . . . 771
7.31 rti::topic Namespace Reference . . . 771
7.31.1 Detailed Description . . . 774
7.31.2 Function Documentation . . . 774
7.31.2.1 find_topics() [1/2] . . . 774
7.31.2.2 find_topics() [2/2] . . . 775
7.31.2.3 find_registered_content_filters() [1/2] . . . 775
7.31.2.4 find_registered_content_filters() [2/2] . . . 776
7.32 rti::util Namespace Reference . . . 777
7.32.1 Detailed Description . . . 777
7.33 rti::util::activation Namespace Reference . . . 778
7.33.1 Detailed Description . . . 778
7.34 rti::util::discovery Namespace Reference . . . 778
7.34.1 Detailed Description . . . 778
7.35 rti::util::function_history Namespace Reference . . . 778
7.35.1 Detailed Description . . . 779
7.36 rti::util::heap_monitoring Namespace Reference . . . 779
7.36.1 Detailed Description . . . 779
7.37 rti::util::network_capture Namespace Reference . . . 780
7.37.1 Detailed Description . . . 780

8 Class Documentation **781**

8.1 rti::flat::AbstractAlignedList< ElementOffset > Class Template Reference . . . 781
8.1.1 Detailed Description . . . 781
8.1.2 Member Typedef Documentation . . . 782

8.1.2.1 iterator . . . 782
8.1.3 Member Function Documentation . . . 782
8.1.3.1 begin() . . . 782
8.1.3.2 end() . . . 782
8.2 rti::flat::AbstractBuilder Class Reference . . . 783
8.2.1 Detailed Description . . . 783
8.2.2 Constructor & Destructor Documentation . . . 783
8.2.2.1 ~AbstractBuilder() . . . 783
8.2.3 Member Function Documentation . . . 784
8.2.3.1 discard() . . . 784
8.2.3.2 is_nested() . . . 784
8.2.3.3 is_valid() . . . 784
8.2.3.4 capacity() . . . 785
8.3 dds::core::xtypes::AbstractConstructedType< MemberType > Class Template Reference . . . 785
8.3.1 Detailed Description . . . 786
8.3.2 Member Typedef Documentation . . . 786
8.3.2.1 Member . . . 786
8.3.2.2 MemberIndex . . . 787
8.3.3 Member Function Documentation . . . 787
8.3.3.1 extensibility_kind() . . . 787
8.3.3.2 member_count() . . . 787
8.3.3.3 member() [1/2] . . . 787
8.3.3.4 member() [2/2] . . . 788
8.3.3.5 find_member_by_name() . . . 788
8.3.3.6 members() . . . 788
8.3.3.7 cdr_serialized_sample_max_size() . . . 789
8.3.3.8 cdr_serialized_sample_min_size() . . . 789
8.3.3.9 cdr_serialized_sample_key_max_size() . . . 790
8.3.4 Member Data Documentation . . . 790
8.3.4.1 INVALID_INDEX . . . 790
8.4 rti::flat::AbstractListBuilder< DHeaderGen > Class Template Reference . . . 791
8.4.1 Detailed Description . . . 791
8.4.2 Member Function Documentation . . . 791
8.4.2.1 element_count() . . . 791
8.5 rti::flat::AbstractPrimitiveList< T > Class Template Reference . . . 792
8.5.1 Detailed Description . . . 792
8.5.2 Member Function Documentation . . . 792
8.5.2.1 get_element() . . . 792
8.5.2.2 set_element() . . . 794

8.6 rti::flat::AbstractSequenceBuilder< DHeaderGen > Class Template Reference . . . 794
8.6.1 Detailed Description . . . 795
8.6.2 Member Function Documentation . . . 795
8.6.2.1 element_count() . . . 795
8.7 rti::pub::AcknowledgmentInfo Class Reference . . . 795
8.7.1 Detailed Description . . . 796
8.7.2 Member Function Documentation . . . 796
8.7.2.1 subscription_handle() . . . 796
8.7.2.2 sample_identity() . . . 796
8.7.2.3 valid_response_data() . . . 796
8.7.2.4 response_data() . . . 797
8.8 rti::core::policy::AcknowledgmentKind_def Struct Reference . . . 797
8.8.1 Detailed Description . . . 797
8.8.2 Member Enumeration Documentation . . . 797
8.8.2.1 type . . . 797
8.9 rti::sub::AckResponseData Class Reference . . . 798
8.9.1 Detailed Description . . . 798
8.9.2 Constructor & Destructor Documentation . . . 799
8.9.2.1 AckResponseData() [1/2] . . . 799
8.9.2.2 AckResponseData() [2/2] . . . 799
8.9.3 Member Function Documentation . . . 799
8.9.3.1 value() [1/2] . . . 799
8.9.3.2 value() [2/2] . . . 799
8.9.3.3 begin() . . . 800
8.9.3.4 end() . . . 800
8.10 rti::flat::AggregationBuilder Class Reference . . . 800
8.10.1 Detailed Description . . . 800
8.11 dds::core::xtypes::AliasType Class Reference . . . 801
8.11.1 Detailed Description . . . 801
8.11.2 Constructor & Destructor Documentation . . . 801
8.11.2.1 AliasType() . . . 801
8.11.3 Member Function Documentation . . . 802
8.11.3.1 related_type() . . . 802
8.11.3.2 is_pointer() . . . 802
8.11.3.3 unit() . . . 802
8.11.4 Friends And Related Function Documentation . . . 802
8.11.4.1 resolve_alias() . . . 803
8.12 rti::core::AllocationSettings Class Reference . . . 803
8.12.1 Detailed Description . . . 803

8.12.2 Constructor & Destructor Documentation . . . 804
8.12.2.1 AllocationSettings() [1/2] . . . 804
8.12.2.2 AllocationSettings() [2/2] . . . 804
8.12.3 Member Function Documentation . . . 804
8.12.3.1 initial_count() [1/2] . . . 804
8.12.3.2 initial_count() [2/2] . . . 804
8.12.3.3 max_count() [1/2] . . . 805
8.12.3.4 max_count() [2/2] . . . 805
8.12.3.5 incremental_count() [1/2] . . . 805
8.12.3.6 incremental_count() [2/2] . . . 805
8.12.4 Member Data Documentation . . . 805
8.12.4.1 AUTO_COUNT . . . 806
8.13 dds::core::AlreadyClosedError Class Reference . . . 806
8.13.1 Detailed Description . . . 806
8.13.2 Member Function Documentation . . . 806
8.13.2.1 what() . . . 807
8.14 dds::sub::AnyDataReader Class Reference . . . 807
8.14.1 Detailed Description . . . 808
8.14.2 Constructor & Destructor Documentation . . . 808
8.14.2.1 AnyDataReader() . . . 808
8.14.3 Member Function Documentation . . . 809
8.14.3.1 operator=() [1/2] . . . 809
8.14.3.2 qos() [1/2] . . . 809
8.14.3.3 qos() [2/2] . . . 809
8.14.3.4 topic_name() . . . 810
8.14.3.5 type_name() . . . 810
8.14.3.6 subscriber() . . . 810
8.14.3.7 operator=() [2/2] . . . 810
8.14.3.8 get() . . . 810
8.14.3.9 close() . . . 811
8.14.3.10 retain() . . . 811
8.14.4 Friends And Related Function Documentation . . . 811
8.14.4.1 get() . . . 812
8.15 dds::sub::AnyDataReaderListener Class Reference . . . 812
8.15.1 Detailed Description . . . 813
8.15.2 Member Function Documentation . . . 813
8.15.2.1 on_requested_deadline_missed() . . . 813
8.15.2.2 on_requested_incompatible_qos() . . . 813
8.15.2.3 on_sample_rejected() . . . 814

8.15.2.4 on_liveliness_changed() . . . 814
8.15.2.5 on_data_available() . . . 814
8.15.2.6 on_subscription_matched() . . . 814
8.15.2.7 on_sample_lost() . . . 815
8.16 dds::pub::AnyDataWriter Class Reference . . . 815
8.16.1 Detailed Description . . . 816
8.16.2 Constructor & Destructor Documentation . . . 816
8.16.2.1 AnyDataWriter() . . . 816
8.16.3 Member Function Documentation . . . 817
8.16.3.1 operator=() [1/2] . . . 817
8.16.3.2 qos() [1/2] . . . 817
8.16.3.3 qos() [2/2] . . . 817
8.16.3.4 topic_name() . . . 818
8.16.3.5 type_name() . . . 818
8.16.3.6 publisher() . . . 818
8.16.3.7 wait_for_acknowledgments() . . . 818
8.16.3.8 close() . . . 819
8.16.3.9 retain() . . . 819
8.16.3.10 operator=() [2/2] . . . 819
8.16.3.11 get() . . . 819
8.16.4 Friends And Related Function Documentation . . . 820
8.16.4.1 get() . . . 820
8.17 dds::pub::AnyDataWriterListener Class Reference . . . 821
8.17.1 Detailed Description . . . 822
8.17.2 Member Function Documentation . . . 822
8.17.2.1 on_offered_deadline_missed() . . . 822
8.17.2.2 on_offered_incompatible_qos() . . . 822
8.17.2.3 on_liveliness_lost() . . . 823
8.17.2.4 on_publication_matched() . . . 823
8.17.2.5 on_reliable_writer_cache_changed() . . . 823
8.17.2.6 on_reliable_reader_activity_changed() . . . 823
8.17.2.7 on_sample_removed() . . . 824
8.17.2.8 on_instance_replaced() . . . 824
8.17.2.9 on_application_acknowledgment() . . . 824
8.17.2.10 on_service_request_accepted() . . . 824
8.18 dds::topic::AnyTopic Class Reference . . . 825
8.18.1 Detailed Description . . . 825
8.18.2 Constructor & Destructor Documentation . . . 826
8.18.2.1 AnyTopic() . . . 826

8.18.3 Member Function Documentation . . . 826
8.18.3.1 domain_participant() . . . 826
8.18.3.2 inconsistent_topic_status() . . . 826
8.18.3.3 qos() [1/2] . . . 827
8.18.3.4 qos() [2/2] . . . 827
8.18.3.5 name() . . . 827
8.18.3.6 type_name() . . . 827
8.18.3.7 close() . . . 827
8.18.3.8 get() . . . 827
8.18.4 Friends And Related Function Documentation . . . 828
8.18.4.1 get() . . . 828
8.19 dds::core::xtypes::ArrayType Class Reference . . . 828
8.19.1 Detailed Description . . . 829
8.19.2 Constructor & Destructor Documentation . . . 829
8.19.2.1 ArrayType() [1/6] . . . 829
8.19.2.2 ArrayType() [2/6] . . . 830
8.19.2.3 ArrayType() [3/6] . . . 830
8.19.2.4 ArrayType() [4/6] . . . 831
8.19.2.5 ArrayType() [5/6] . . . 831
8.19.2.6 ArrayType() [6/6] . . . 831
8.19.3 Member Function Documentation . . . 832
8.19.3.1 dimension_count() . . . 832
8.19.3.2 dimension() . . . 832
8.19.3.3 total_element_count() . . . 832
8.20 rti::core::policy::AsynchronousPublisher Class Reference . . . 832
8.20.1 Detailed Description . . . 834
8.20.2 Usage . . . 834
8.20.3 Constructor & Destructor Documentation . . . 835
8.20.3.1 AsynchronousPublisher() . . . 835
8.20.4 Member Function Documentation . . . 835
8.20.4.1 disable_asynchronous_write() [1/2] . . . 835
8.20.4.2 disable_asynchronous_write() [2/2] . . . 835
8.20.4.3 thread() [1/3] . . . 836
8.20.4.4 thread() [2/3] . . . 836
8.20.4.5 thread() [3/3] . . . 836
8.20.4.6 disable_asynchronous_batch() [1/2] . . . 837
8.20.4.7 disable_asynchronous_batch() [2/2] . . . 837
8.20.4.8 asynchronous_batch_thread() [1/3] . . . 837
8.20.4.9 asynchronous_batch_thread() [2/3] . . . 838

8.20.4.10 asynchronous_batch_thread() [3/3] . . . 838
8.20.4.11 disable_topic_query_publication() [1/2] . . . 838
8.20.4.12 disable_topic_query_publication() [2/2] . . . 838
8.20.4.13 topic_query_publication_thread() [1/3] . . . 839
8.20.4.14 topic_query_publication_thread() [2/3] . . . 839
8.20.4.15 topic_query_publication_thread() [3/3] . . . 839
8.20.4.16 Enabled() . . . 840
8.20.4.17 Disabled() . . . 840
8.21 rti::core::cond::AsyncWaitSet Class Reference . . . 840
8.21.1 Detailed Description . . . 841
8.21.2 AsyncWaitSet Thread Orchestration . . . 842
8.21.3 AsyncWaitSet Thread Safety . . . 842
8.21.3.1 Condition Locking . . . 843
8.21.4 AsyncWaitSet Events and Resources . . . 843
8.21.5 Constructor & Destructor Documentation . . . 844
8.21.5.1 AsyncWaitSet() [1/3] . . . 844
8.21.5.2 AsyncWaitSet() [2/3] . . . 844
8.21.5.3 AsyncWaitSet() [3/3] . . . 845
8.21.6 Member Function Documentation . . . 845
8.21.6.1 start() [1/2] . . . 845
8.21.6.2 start() [2/2] . . . 846
8.21.6.3 stop() [1/2] . . . 846
8.21.6.4 stop() [2/2] . . . 847
8.21.6.5 attach_condition() [1/2] . . . 848
8.21.6.6 attach_condition() [2/2] . . . 848
8.21.6.7 detach_condition() [1/2] . . . 849
8.21.6.8 detach_condition() [2/2] . . . 850
8.21.6.9 operator+=() . . . 851
8.21.6.10 operator-=() . . . 851
8.21.6.11 unlock_condition() . . . 852
8.21.6.12 property() . . . 852
8.21.6.13 conditions() [1/2] . . . 853
8.21.6.14 conditions() [2/2] . . . 853
8.22 rti::core::cond::AsyncWaitSetCompletionToken Class Reference . . . 853
8.22.1 Detailed Description . . . 854
8.22.2 AsyncWaitSetCompletionToken management . . . 855
8.22.3 Constructor & Destructor Documentation . . . 855
8.22.3.1 AsyncWaitSetCompletionToken() . . . 855
8.22.4 Member Function Documentation . . . 855

8.22.4.1 wait() . . . 856
8.23 rti::core::cond::AsyncWaitSetListener Class Reference . . . 856
8.23.1 Detailed Description . . . 857
8.23.2 Member Function Documentation . . . 857
8.23.2.1 on_thread_spawned() . . . 857
8.23.2.2 on_thread_deleted() . . . 857
8.23.2.3 on_wait_timeout() . . . 857
8.24 rti::core::cond::AsyncWaitSetProperty Class Reference . . . 858
8.24.1 Detailed Description . . . 859
8.24.2 Constructor & Destructor Documentation . . . 859
8.24.2.1 AsyncWaitSetProperty() [1/2] . . . 859
8.24.2.2 AsyncWaitSetProperty() [2/2] . . . 859
8.24.3 Member Function Documentation . . . 859
8.24.3.1 waitset_property() [1/2] . . . 859
8.24.3.2 waitset_property() [2/2] . . . 860
8.24.3.3 wait_timeout() [1/2] . . . 860
8.24.3.4 wait_timeout() [2/2] . . . 860
8.24.3.5 level() [1/2] . . . 860
8.24.3.6 level() [2/2] . . . 861
8.24.3.7 thread_name_prefix() [1/2] . . . 861
8.24.3.8 thread_name_prefix() [2/2] . . . 861
8.24.3.9 thread_settings() [1/2] . . . 861
8.24.3.10 thread_settings() [2/2] . . . 862
8.24.3.11 thread_pool_size() [1/2] . . . 862
8.24.3.12 thread_pool_size() [2/2] . . . 862
8.25 rti::util::atomic_shared_ptr< T > Class Template Reference . . . 862
8.25.1 Detailed Description . . . 863
8.26 rti::config::activity_context::AttributeKindMask Class Reference . . . 863
8.26.1 Detailed Description . . . 864
8.26.2 Member Typedef Documentation . . . 864
8.26.2.1 MaskType . . . 864
8.26.3 Constructor & Destructor Documentation . . . 864
8.26.3.1 AttributeKindMask() [1/3] . . . 864
8.26.3.2 AttributeKindMask() [2/3] . . . 864
8.26.3.3 AttributeKindMask() [3/3] . . . 865
8.26.4 Member Function Documentation . . . 865
8.26.4.1 guid_prefix() . . . 865
8.26.4.2 topic() . . . 865
8.26.4.3 type() . . . 866

8.26.4.4 entity_kind() . . . 866
8.26.4.5 domain_id() . . . 866
8.26.4.6 entity_name() . . . 867
8.26.4.7 default_mask() . . . 867
8.26.4.8 none() . . . 867
8.26.4.9 all() . . . 867
8.27 rti::core::policy::Availability Class Reference . . . 868
8.27.1 Detailed Description . . . 868
8.27.2 Usage . . . 869
8.27.3 Consistency . . . 870
8.27.4 Constructor & Destructor Documentation . . . 870
8.27.4.1 Availability() [1/2] . . . 870
8.27.4.2 Availability() [2/2] . . . 871
8.27.5 Member Function Documentation . . . 871
8.27.5.1 enable_required_subscriptions() [1/2] . . . 871
8.27.5.2 enable_required_subscriptions() [2/2] . . . 871
8.27.5.3 max_data_availability_waiting_time() [1/2] . . . 872
8.27.5.4 max_data_availability_waiting_time() [2/2] . . . 872
8.27.5.5 max_endpoint_availability_waiting_time() [1/2] . . . 873
8.27.5.6 max_endpoint_availability_waiting_time() [2/2] . . . 873
8.27.5.7 required_matched_endpoint_groups() [1/2] . . . 874
8.27.5.8 required_matched_endpoint_groups() [2/2] . . . 874
8.28 dds::core::basic_string< CharType, Allocator > Class Template Reference . . . 875
8.28.1 Detailed Description . . . 876
8.28.2 Constructor & Destructor Documentation . . . 876
8.28.2.1 basic_string() [1/6] . . . 876
8.28.2.2 basic_string() [2/6] . . . 876
8.28.2.3 basic_string() [3/6] . . . 876
8.28.2.4 basic_string() [4/6] . . . 877
8.28.2.5 basic_string() [5/6] . . . 877
8.28.2.6 basic_string() [6/6] . . . 877
8.28.3 Member Function Documentation . . . 877
8.28.3.1 c_str() . . . 877
8.28.3.2 size() . . . 878
8.28.3.3 operator=() [1/2] . . . 878
8.28.3.4 operator=() [2/2] . . . 878
8.28.3.5 operator==() . . . 878
8.28.3.6 operator"!=() . . . 879
8.28.3.7 to_std_string() . . . 879

8.28.3.8 operator std::basic_string< CharType >() . . . 879
8.28.4 Friends And Related Function Documentation . . . 879
8.28.4.1 operator<<() . . . 879
8.29 rti::core::policy::Batch Class Reference . . . 880
8.29.1 Detailed Description . . . 881
8.29.2 Constructor & Destructor Documentation . . . 881
8.29.2.1 Batch() . . . 881
8.29.3 Member Function Documentation . . . 881
8.29.3.1 Enabled() . . . 881
8.29.3.2 Disabled() . . . 882
8.29.3.3 EnabledWithMaxDataBytes() . . . 882
8.29.3.4 EnabledWithMaxSamples() . . . 882
8.29.3.5 enable() [1/2] . . . 882
8.29.3.6 enable() [2/2] . . . 882
8.29.3.7 max_data_bytes() [1/2] . . . 883
8.29.4 Consistency . . . 883
8.29.4.1 max_data_bytes() [2/2] . . . 883
8.29.4.2 max_samples() [1/2] . . . 884
8.29.5 Consistency . . . 884
8.29.5.1 max_samples() [2/2] . . . 884
8.29.5.2 max_flush_delay() [1/2] . . . 884
8.29.6 Consistency . . . 885
8.29.6.1 max_flush_delay() [2/2] . . . 885
8.29.6.2 source_timestamp_resolution() [1/2] . . . 885
8.29.7 Consistency . . . 886
8.29.7.1 source_timestamp_resolution() [2/2] . . . 886
8.29.7.2 thread_safe_write() [1/2] . . . 886
8.29.8 Consistency . . . 886
8.29.8.1 thread_safe_write() [2/2] . . . 887
8.30 rti::core::bounded_sequence< T, MaxLength > Class Template Reference . . . 887
8.30.1 Detailed Description . . . 890
8.30.2 Member Typedef Documentation . . . 891
8.30.2.1 vector_type . . . 891
8.30.2.2 value_type . . . 891
8.30.2.3 allocator_type . . . 891
8.30.2.4 size_type . . . 891
8.30.2.5 difference_type . . . 891
8.30.2.6 reference . . . 892
8.30.2.7 const_reference . . . 892

8.30.2.8 pointer . . . 892
8.30.2.9 const_pointer . . . 892
8.30.2.10 iterator . . . 892
8.30.2.11 const_iterator . . . 893
8.30.2.12 reverse_iterator . . . 893
8.30.2.13 const_reverse_iterator . . . 893
8.30.3 Constructor & Destructor Documentation . . . 893
8.30.3.1 bounded_sequence() [1/9] . . . 893
8.30.3.2 bounded_sequence() [2/9] . . . 893
8.30.3.3 bounded_sequence() [3/9] . . . 894
8.30.3.4 bounded_sequence() [4/9] . . . 894
8.30.3.5 bounded_sequence() [5/9] . . . 894
8.30.3.6 bounded_sequence() [6/9] . . . 894
8.30.3.7 bounded_sequence() [7/9] . . . 895
8.30.3.8 bounded_sequence() [8/9] . . . 895
8.30.3.9 bounded_sequence() [9/9] . . . 895
8.30.4 Member Function Documentation . . . 895
8.30.4.1 operator=() [1/6] . . . 895
8.30.4.2 operator=() [2/6] . . . 896
8.30.4.3 operator=() [3/6] . . . 896
8.30.4.4 operator=() [4/6] . . . 896
8.30.4.5 operator=() [5/6] . . . 896
8.30.4.6 operator=() [6/6] . . . 896
8.30.4.7 operator[]() [1/2] . . . 897
8.30.4.8 operator[]() [2/2] . . . 897
8.30.4.9 at() [1/2] . . . 897
8.30.4.10 at() [2/2] . . . 897
8.30.4.11 front() [1/2] . . . 898
8.30.4.12 front() [2/2] . . . 898
8.30.4.13 back() [1/2] . . . 898
8.30.4.14 back() [2/2] . . . 898
8.30.4.15 data() [1/2] . . . 899
8.30.4.16 data() [2/2] . . . 899
8.30.4.17 begin() [1/2] . . . 899
8.30.4.18 begin() [2/2] . . . 899
8.30.4.19 cbegin() . . . 899
8.30.4.20 end() [1/2] . . . 900
8.30.4.21 end() [2/2] . . . 900
8.30.4.22 cend() . . . 900

8.30.4.23 rbegin() [1/2] . . . 900
8.30.4.24 rbegin() [2/2] . . . 900
8.30.4.25 crbegin() . . . 901
8.30.4.26 rend() . . . 901
8.30.4.27 crend() . . . 901
8.30.4.28 empty() . . . 901
8.30.4.29 size() . . . 901
8.30.4.30 max_size() . . . 902
8.30.4.31 reserve() . . . 902
8.30.4.32 capacity() . . . 902
8.30.4.33 clear() . . . 902
8.30.4.34 shrink_to_fit() . . . 903
8.30.4.35 insert() . . . 903
8.30.4.36 erase() . . . 903
8.30.4.37 push_back() [1/2] . . . 903
8.30.4.38 push_back() [2/2] . . . 904
8.30.4.39 pop_back() . . . 904
8.30.4.40 resize() [1/2] . . . 904
8.30.4.41 resize() [2/2] . . . 905
8.30.4.42 swap() . . . 905
8.31 dds::topic::BuiltinTopicKey Class Reference . . . 905
8.31.1 Detailed Description . . . 906
8.31.2 Member Function Documentation . . . 906
8.31.2.1 TBuiltinTopicKey() . . . 906
8.31.2.2 unknown() . . . 906
8.31.2.3 value() . . . 906
8.32 rti::core::policy::BuiltinTopicReaderResourceLimits Class Reference . . . 906
8.32.1 Detailed Description . . . 908
8.32.2 Constructor & Destructor Documentation . . . 908
8.32.2.1 BuiltinTopicReaderResourceLimits() . . . 909
8.32.3 Member Function Documentation . . . 909
8.32.3.1 initial_samples() [1/2] . . . 909
8.32.3.2 initial_samples() [2/2] . . . 909
8.32.3.3 max_samples() [1/2] . . . 910
8.32.3.4 max_samples() [2/2] . . . 910
8.32.3.5 initial_infos() [1/2] . . . 910
8.32.3.6 initial_infos() [2/2] . . . 911
8.32.3.7 max_infos() [1/2] . . . 911
8.32.3.8 max_infos() [2/2] . . . 911

8.32.3.9 initial_outstanding_reads() [1/2] . . . 911
8.32.3.10 initial_outstanding_reads() [2/2] . . . 912
8.32.3.11 max_outstanding_reads() [1/2] . . . 912
8.32.3.12 max_outstanding_reads() [2/2] . . . 912
8.32.3.13 max_samples_per_read() [1/2] . . . 912
8.32.3.14 max_samples_per_read() [2/2] . . . 913
8.32.3.15 disable_fragmentation_support() [1/2] . . . 913
8.32.3.16 disable_fragmentation_support() [2/2] . . . 913
8.32.3.17 max_fragmented_samples() [1/2] . . . 914
8.32.3.18 max_fragmented_samples() [2/2] . . . 914
8.32.3.19 initial_fragmented_samples() [1/2] . . . 915
8.32.3.20 initial_fragmented_samples() [2/2] . . . 915
8.32.3.21 max_fragmented_samples_per_remote_writer() [1/2] . . . 915
8.32.3.22 max_fragmented_samples_per_remote_writer() [2/2] . . . 916
8.32.3.23 max_fragments_per_sample() [1/2] . . . 916
8.32.3.24 max_fragments_per_sample() [2/2] . . . 916
8.32.3.25 dynamically_allocate_fragmented_samples() [1/2] . . . 917
8.32.3.26 dynamically_allocate_fragmented_samples() [2/2] . . . 917
8.33 dds::core::BytesTopicType Class Reference . . . 917
8.33.1 Detailed Description . . . 918
8.33.2 Constructor & Destructor Documentation . . . 918
8.33.2.1 BytesTopicType() [1/2] . . . 918
8.33.2.2 BytesTopicType() [2/2] . . . 918
8.33.3 Member Function Documentation . . . 919
8.33.3.1 operator std::vector< uint8_t >() . . . 919
8.33.3.2 data() [1/2] . . . 919
8.33.3.3 data() [2/2] . . . 919
8.33.3.4 operator[]() [1/2] . . . 920
8.33.3.5 operator[]() [2/2] . . . 920
8.33.3.6 length() . . . 920
8.34 rti::core::policy::CdrPaddingKind_def Struct Reference . . . 920
8.34.1 Detailed Description . . . 921
8.34.2 Member Enumeration Documentation . . . 921
8.34.2.1 type . . . 921
8.35 rti::core::ChannelSettings Class Reference . . . 921
8.35.1 Detailed Description . . . 922
8.35.2 Constructor & Destructor Documentation . . . 922
8.35.2.1 ChannelSettings() . . . 922
8.35.3 Member Function Documentation . . . 922

8.35.3.1 multicast_settings() [1/2] . . . 923
8.35.3.2 multicast_settings() [2/2] . . . 923
8.35.3.3 filter_expression() [1/2] . . . 923
8.35.3.4 filter_expression() [2/2] . . . 924
8.35.3.5 priority() [1/2] . . . 924
8.35.3.6 priority() [2/2] . . . 924
8.36 dds::rpc::ClientEndpoint< Request, Reply > Class Template Reference . . . 925
8.36.1 Detailed Description . . . 925
8.36.2 Member Typedef Documentation . . . 926
8.36.2.1 RequestType . . . 926
8.36.2.2 ReplyType . . . 926
8.36.3 Constructor & Destructor Documentation . . . 926
8.36.3.1 ClientEndpoint() . . . 926
8.36.4 Member Function Documentation . . . 926
8.36.4.1 close() . . . 926
8.36.4.2 closed() . . . 927
8.36.4.3 wait_for_service() . . . 927
8.36.4.4 request_datawriter() . . . 927
8.36.4.5 reply_datareader() . . . 928
8.37 dds::rpc::ClientParams Class Reference . . . 928
8.37.1 Detailed Description . . . 928
8.37.2 Constructor & Destructor Documentation . . . 928
8.37.2.1 ClientParams() . . . 928
8.37.3 Member Function Documentation . . . 929
8.37.3.1 function_call_max_wait() [1/2] . . . 929
8.37.3.2 function_call_max_wait() [2/2] . . . 929
8.38 dds::sub::CoherentAccess Class Reference . . . 929
8.38.1 Detailed Description . . . 930
8.38.2 Constructor & Destructor Documentation . . . 930
8.38.2.1 CoherentAccess() . . . 930
8.38.2.2 ~CoherentAccess() . . . 931
8.38.3 Member Function Documentation . . . 931
8.38.3.1 end() . . . 931
8.39 dds::pub::CoherentSet Class Reference . . . 932
8.39.1 Detailed Description . . . 932
8.39.2 Constructor & Destructor Documentation . . . 932
8.39.2.1 CoherentSet() . . . 932
8.39.2.2 ~CoherentSet() . . . 933
8.39.3 Member Function Documentation . . . 933

8.39.3.1 end() . . . 934
8.40 rti::core::CoherentSetInfo Class Reference . . . 934
8.40.1 Detailed Description . . . 935
8.40.2 Constructor & Destructor Documentation . . . 935
8.40.2.1 CoherentSetInfo() [1/2] . . . 935
8.40.2.2 CoherentSetInfo() [2/2] . . . 936
8.40.3 Member Function Documentation . . . 936
8.40.3.1 group_guid() [1/3] . . . 936
8.40.3.2 group_guid() [2/3] . . . 936
8.40.3.3 group_guid() [3/3] . . . 936
8.40.3.4 coherent_set_sequence_number() [1/3] . . . 937
8.40.3.5 coherent_set_sequence_number() [2/3] . . . 937
8.40.3.6 coherent_set_sequence_number() [3/3] . . . 937
8.40.3.7 group_coherent_set_sequence_number() [1/3] . . . 937
8.40.3.8 group_coherent_set_sequence_number() [2/3] . . . 938
8.40.3.9 group_coherent_set_sequence_number() [3/3] . . . 938
8.40.3.10 incomplete_coherent_set() [1/2] . . . 938
8.40.3.11 incomplete_coherent_set() [2/2] . . . 938
8.40.3.12 unknown() . . . 938
8.40.4 Friends And Related Function Documentation . . . 939
8.40.4.1 operator<<() . . . 939
8.41 dds::core::xtypes::CollectionType Class Reference . . . 939
8.41.1 Detailed Description . . . 939
8.41.2 Member Function Documentation . . . 939
8.41.2.1 content_type() . . . 940
8.42 rti::core::CompressionIdMask Class Reference . . . 940
8.42.1 Detailed Description . . . 941
8.42.2 Member Typedef Documentation . . . 941
8.42.2.1 MaskType . . . 941
8.42.3 Constructor & Destructor Documentation . . . 941
8.42.3.1 CompressionIdMask() [1/2] . . . 941
8.42.3.2 CompressionIdMask() [2/2] . . . 941
8.42.4 Member Function Documentation . . . 941
8.42.4.1 all() . . . 942
8.42.4.2 none() . . . 942
8.42.4.3 default_publication() . . . 942
8.42.4.4 default_subscription() . . . 942
8.42.4.5 zlib() . . . 942
8.42.4.6 bzip2() . . . 943

8.42.4.7 lz4() . . . 943
8.43 rti::core::CompressionSettings Class Reference . . . 943
8.43.1 Detailed Description . . . 944
8.43.2 Constructor & Destructor Documentation . . . 944
8.43.2.1 CompressionSettings() [1/3] . . . 944
8.43.2.2 CompressionSettings() [2/3] . . . 944
8.43.2.3 CompressionSettings() [3/3] . . . 944
8.43.3 Member Function Documentation . . . 945
8.43.3.1 compression_level_default() . . . 945
8.43.3.2 compression_level_best_compression() . . . 945
8.43.3.3 compression_level_best_speed() . . . 945
8.43.3.4 compression_ids() [1/2] . . . 945
8.43.3.5 compression_ids() [2/2] . . . 946
8.43.3.6 writer_compression_level() [1/2] . . . 946
8.43.3.7 writer_compression_level() [2/2] . . . 947
8.43.3.8 writer_compression_threshold() [1/2] . . . 947
8.43.3.9 writer_compression_threshold() [2/2] . . . 948
8.44 dds::core::cond::Condition Class Reference . . . 948
8.44.1 Detailed Description . . . 949
8.44.2 Member Function Documentation . . . 949
8.44.2.1 dispatch() . . . 949
8.44.2.2 trigger_value() . . . 949
8.45 rti::queuing::ConsumerAvailabilityParams Class Reference . . . 949
8.45.1 Detailed Description . . . 950
8.45.2 Member Data Documentation . . . 950
8.45.2.1 reception_enabled . . . 950
8.45.2.2 unacked_threshold . . . 950
8.46 rti::topic::ContentFilter< T, CompileData > Class Template Reference . . . 950
8.46.1 Detailed Description . . . 951
8.46.2 Member Function Documentation . . . 952
8.46.2.1 compile() . . . 952
8.46.2.2 evaluate() . . . 953
8.46.2.3 finalize() . . . 953
8.47 dds::topic::ContentFilteredTopic< T > Class Template Reference . . . 954
8.47.1 Detailed Description . . . 955
8.47.2 Constructor & Destructor Documentation . . . 955
8.47.2.1 ContentFilteredTopic() . . . 956
8.47.3 Member Function Documentation . . . 956
8.47.3.1 filter_expression() . . . 956

8.47.3.2 filter_parameters() [1/2] . . . 957
8.47.3.3 filter_parameters() [2/2] . . . 957
8.47.3.4 topic() . . . 957
8.47.3.5 filter() . . . 957
8.47.3.6 append_to_expression_parameter() . . . 958
8.47.3.7 remove_from_expression_parameter() . . . 959
8.47.4 Friends And Related Function Documentation . . . 959
8.47.4.1 find_registered_content_filters() . . . 959
8.48 rti::core::ContentFilterProperty Class Reference . . . 960
8.48.1 Detailed Description . . . 961
8.48.2 Member Function Documentation . . . 961
8.48.2.1 content_filter_topic_name() . . . 961
8.48.2.2 expression_parameters() . . . 961
8.48.2.3 filter_class_name() . . . 961
8.48.2.4 filter_expression() . . . 961
8.48.2.5 related_topic_name() . . . 961
8.49 rti::util::network_capture::ContentKindMask Class Reference . . . 962
8.49.1 Detailed Description . . . 962
8.49.2 Member Typedef Documentation . . . 963
8.49.2.1 MaskType . . . 963
8.49.3 Constructor & Destructor Documentation . . . 963
8.49.3.1 ContentKindMask() [1/3] . . . 963
8.49.3.2 ContentKindMask() [2/3] . . . 963
8.49.3.3 ContentKindMask() [3/3] . . . 963
8.49.4 Member Function Documentation . . . 964
8.49.4.1 user() . . . 964
8.49.4.2 encrypted() . . . 964
8.49.4.3 default_mask() . . . 964
8.49.4.4 none() . . . 964
8.49.4.5 all() . . . 965
8.50 rti::core::Cookie Class Reference . . . 965
8.50.1 Detailed Description . . . 965
8.50.2 Constructor & Destructor Documentation . . . 966
8.50.2.1 Cookie() [1/2] . . . 966
8.50.2.2 Cookie() [2/2] . . . 966
8.50.3 Member Function Documentation . . . 966
8.50.3.1 value() [1/2] . . . 966
8.50.3.2 value() [2/2] . . . 966
8.50.3.3 to_pointer() . . . 966

8.50.4 Friends And Related Function Documentation . . . 967
8.50.4.1 CookieSeq . . . 967
8.50.4.2 operator<<() . . . 967
8.51 rti::topic::CustomFilter< T > Class Template Reference . . . 967
8.51.1 Detailed Description . . . 968
8.51.2 Constructor & Destructor Documentation . . . 968
8.51.2.1 CustomFilter() . . . 968
8.51.3 Member Function Documentation . . . 969
8.51.3.1 get() [1/2] . . . 969
8.51.3.2 get() [2/2] . . . 969
8.52 rti::core::policy::Database Class Reference . . . 969
8.52.1 Detailed Description . . . 970
8.52.2 Member Function Documentation . . . 971
8.52.2.1 thread() [1/3] . . . 971
8.52.2.2 thread() [2/3] . . . 972
8.52.2.3 thread() [3/3] . . . 972
8.52.2.4 shutdown_timeout() [1/2] . . . 972
8.52.2.5 shutdown_timeout() [2/2] . . . 972
8.52.2.6 cleanup_period() [1/2] . . . 973
8.52.2.7 cleanup_period() [2/2] . . . 973
8.52.2.8 shutdown_cleanup_period() [1/2] . . . 973
8.52.2.9 shutdown_cleanup_period() [2/2] . . . 974
8.52.2.10 initial_records() [1/2] . . . 974
8.52.2.11 initial_records() [2/2] . . . 974
8.52.2.12 max_skiplist_level() . . . 974
8.52.2.13 max_weak_references() [1/2] . . . 975
8.52.2.14 max_weak_references() [2/2] . . . 975
8.52.2.15 initial_weak_references() [1/2] . . . 976
8.52.2.16 initial_weak_references() [2/2] . . . 976
8.53 dds::sub::DataReader< T > Class Template Reference . . . 976
8.53.1 Detailed Description . . . 982
8.53.2 Constructor & Destructor Documentation . . . 983
8.53.2.1 DataReader() [1/12] . . . 983
8.53.2.2 DataReader() [2/12] . . . 984
8.53.2.3 DataReader() [3/12] . . . 985
8.53.2.4 DataReader() [4/12] . . . 986
8.53.2.5 DataReader() [5/12] . . . 986
8.53.2.6 DataReader() [6/12] . . . 987
8.53.2.7 DataReader() [7/12] . . . 987

8.53.2.8 DataReader() [8/12] . . . 988
8.53.2.9 DataReader() [9/12] . . . 988
8.53.2.10 DataReader() [10/12] . . . 989
8.53.2.11 DataReader() [11/12] . . . 989
8.53.2.12 DataReader() [12/12] . . . 991
8.53.3 Member Function Documentation . . . 991
8.53.3.1 default_filter_state() [1/2] . . . 991
8.53.3.2 default_filter_state() [2/2] . . . 992
8.53.3.3 operator>>() [1/4] . . . 993
8.53.3.4 operator>>() [2/4] . . . 993
8.53.3.5 operator>>() [3/4] . . . 995
8.53.3.6 read() [1/5] . . . 996
8.53.3.7 take() [1/5] . . . 997
8.53.3.8 read() [2/5] . . . 1000
8.53.3.9 take() [2/5] . . . 1001
8.53.3.10 read() [3/5] . . . 1001
8.53.3.11 take() [3/5] . . . 1002
8.53.3.12 select() . . . 1003
8.53.3.13 key_value() [1/2] . . . 1004
8.53.3.14 key_value() [2/2] . . . 1004
8.53.3.15 lookup_instance() . . . 1005
8.53.3.16 topic_description() . . . 1005
8.53.3.17 subscriber() . . . 1006
8.53.3.18 listener() [1/2] . . . 1006
8.53.3.19 listener() [2/2] . . . 1006
8.53.3.20 set_listener() [1/2] . . . 1006
8.53.3.21 set_listener() [2/2] . . . 1007
8.53.3.22 get_listener() . . . 1007
8.53.3.23 qos() [1/2] . . . 1008
8.53.3.24 qos() [2/2] . . . 1008
8.53.3.25 operator<<() . . . 1008
8.53.3.26 operator>>() [4/4] . . . 1009
8.53.3.27 wait_for_historical_data() . . . 1009
8.53.3.28 liveliness_changed_status() . . . 1011
8.53.3.29 sample_rejected_status() . . . 1011
8.53.3.30 sample_lost_status() . . . 1011
8.53.3.31 requested_deadline_missed_status() . . . 1012
8.53.3.32 requested_incompatible_qos_status() . . . 1012
8.53.3.33 subscription_matched_status() . . . 1012

8.53.3.34 datareader_cache_status() . . . 1013
8.53.3.35 datareader_protocol_status() . . . 1013
8.53.3.36 matched_publication_datareader_protocol_status() . . . 1013
8.53.3.37 acknowledge_all() [1/2] . . . 1014
8.53.3.38 acknowledge_all() [2/2] . . . 1014
8.53.3.39 acknowledge_sample() [1/2] . . . 1015
8.53.3.40 acknowledge_sample() [2/2] . . . 1015
8.53.3.41 close_contained_entities() . . . 1016
8.53.3.42 read_noexcept() . . . 1018
8.53.3.43 take_noexcept() . . . 1018
8.53.3.44 read() [4/5] . . . 1019
8.53.3.45 read() [5/5] . . . 1019
8.53.3.46 take() [4/5] . . . 1020
8.53.3.47 take() [5/5] . . . 1021
8.53.3.48 is_data_consistent() [1/2] . . . 1022
8.53.3.49 is_data_consistent() [2/2] . . . 1022
8.53.3.50 topic_name() . . . 1023
8.53.3.51 type_name() . . . 1024
8.53.3.52 close() . . . 1024
8.53.4 Friends And Related Function Documentation . . . 1024
8.53.4.1 read() . . . 1024
8.53.4.2 take() . . . 1025
8.53.4.3 max_samples() . . . 1025
8.53.4.4 content() . . . 1026
8.53.4.5 condition() . . . 1026
8.53.4.6 state() . . . 1027
8.53.4.7 instance() . . . 1028
8.53.4.8 next_instance() . . . 1028
8.53.4.9 ignore() [1/2] . . . 1030
8.53.4.10 ignore() [2/2] . . . 1031
8.53.4.11 matched_publications() [1/2] . . . 1031
8.53.4.12 matched_publications() [2/2] . . . 1032
8.53.4.13 matched_publication_data() [1/2] . . . 1034
8.53.4.14 matched_publication_participant_data() . . . 1034
8.53.4.15 is_matched_publication_alive() . . . 1035
8.53.4.16 matched_publication_data() [2/2] . . . 1036
8.53.4.17 find() [1/4] . . . 1037
8.53.4.18 find() [2/4] . . . 1038
8.53.4.19 find() [3/4] . . . 1039

8.53.4.20 find() [4/4] . . . 1040
8.53.4.21 find_datareaders() [1/2] . . . 1040
8.53.4.22 find_datareaders() [2/2] . . . 1041
8.53.4.23 find_datareader_by_topic_name() . . . 1042
8.53.4.24 find_datareader_by_name() [1/2] . . . 1043
8.53.4.25 find_datareader_by_topic_description() . . . 1044
8.53.4.26 find_datareader_by_name() [2/2] . . . 1045
8.54 rti::core::status::DataReaderCacheStatus Class Reference . . . 1046
8.54.1 Detailed Description . . . 1048
8.54.2 Member Function Documentation . . . 1048
8.54.2.1 sample_count() . . . 1048
8.54.2.2 sample_count_peak() . . . 1048
8.54.2.3 old_source_timestamp_dropped_sample_count() . . . 1048
8.54.2.4 tolerance_source_timestamp_dropped_sample_count() . . . 1049
8.54.2.5 ownership_dropped_sample_count() . . . 1049
8.54.2.6 content_filter_dropped_sample_count() . . . 1049
8.54.2.7 time_based_filter_dropped_sample_count() . . . 1049
8.54.2.8 expired_dropped_sample_count() . . . 1050
8.54.2.9 virtual_duplicate_dropped_sample_count() . . . 1050
8.54.2.10 replaced_dropped_sample_count() . . . 1050
8.54.2.11 writer_removed_batch_sample_dropped_sample_count() . . . 1051
8.54.2.12 total_samples_dropped_by_instance_replacement() . . . 1051
8.54.2.13 alive_instance_count() . . . 1051
8.54.2.14 alive_instance_count_peak() . . . 1051
8.54.2.15 no_writers_instance_count() . . . 1052
8.54.2.16 no_writers_instance_count_peak() . . . 1052
8.54.2.17 disposed_instance_count() . . . 1052
8.54.2.18 disposed_instance_count_peak() . . . 1052
8.54.2.19 detached_instance_count() . . . 1053
8.54.2.20 detached_instance_count_peak() . . . 1053
8.54.2.21 compressed_sample_count() . . . 1053
8.55 rti::core::policy::DataReaderInstanceRemovalKind_def Struct Reference . . . 1053
8.55.1 Detailed Description . . . 1054
8.55.2 Member Enumeration Documentation . . . 1054
8.55.2.1 type . . . 1054
8.56 dds::sub::DataReaderListener< T > Class Template Reference . . . 1055
8.56.1 Detailed Description . . . 1056
8.56.2 Member Function Documentation . . . 1057
8.56.2.1 on_requested_deadline_missed() . . . 1057

8.56.2.2 on_requested_incompatible_qos() . . . 1057
8.56.2.3 on_sample_rejected() . . . 1057
8.56.2.4 on_liveliness_changed() . . . 1058
8.56.2.5 on_data_available() . . . 1058
8.56.2.6 on_subscription_matched() . . . 1058
8.56.2.7 on_sample_lost() . . . 1058
8.57 rti::core::policy::DataReaderProtocol Class Reference . . . 1059
8.57.1 Detailed Description . . . 1060
8.57.2 Constructor & Destructor Documentation . . . 1060
8.57.2.1 DataReaderProtocol() . . . 1060
8.57.3 Member Function Documentation . . . 1060
8.57.3.1 virtual_guid() [1/2] . . . 1061
8.57.3.2 virtual_guid() [2/2] . . . 1061
8.57.3.3 rtps_object_id() [1/2] . . . 1061
8.57.3.4 rtps_object_id() [2/2] . . . 1062
8.57.3.5 expects_inline_qos() [1/2] . . . 1062
8.57.3.6 expects_inline_qos() [2/2] . . . 1062
8.57.3.7 disable_positive_acks() [1/2] . . . 1063
8.57.3.8 disable_positive_acks() [2/2] . . . 1063
8.57.3.9 propagate_dispose_of_unregistered_instances() [1/2] . . . 1063
8.57.3.10 propagate_dispose_of_unregistered_instances() [2/2] . . . 1064
8.57.3.11 propagate_unregister_of_disposed_instances() [1/2] . . . 1064
8.57.3.12 propagate_unregister_of_disposed_instances() [2/2] . . . 1064
8.57.3.13 rtps_reliable_reader() [1/3] . . . 1064
8.57.3.14 rtps_reliable_reader() [2/3] . . . 1065
8.57.3.15 rtps_reliable_reader() [3/3] . . . 1065
8.58 rti::core::status::DataReaderProtocolStatus Class Reference . . . 1065
8.58.1 Detailed Description . . . 1066
8.58.2 Member Function Documentation . . . 1066
8.58.2.1 received_sample_count() . . . 1067
8.58.2.2 received_sample_bytes() . . . 1067
8.58.2.3 duplicate_sample_count() . . . 1068
8.58.2.4 duplicate_sample_bytes() . . . 1068
8.58.2.5 filtered_sample_count() . . . 1068
8.58.2.6 filtered_sample_bytes() . . . 1068
8.58.2.7 received_heartbeat_count() . . . 1068
8.58.2.8 received_heartbeat_bytes() . . . 1069
8.58.2.9 sent_ack_count() . . . 1069
8.58.2.10 sent_ack_bytes() . . . 1069

8.58.2.11 sent_nack_count() . . . 1069
8.58.2.12 sent_nack_bytes() . . . 1069
8.58.2.13 received_gap_count() . . . 1069
8.58.2.14 received_gap_bytes() . . . 1070
8.58.2.15 rejected_sample_count() . . . 1070
8.58.2.16 first_available_sample_sequence_number() . . . 1070
8.58.2.17 last_available_sample_sequence_number() . . . 1070
8.58.2.18 last_committed_sample_sequence_number() . . . 1071
8.58.2.19 uncommitted_sample_count() . . . 1071
8.58.2.20 received_fragment_count() . . . 1071
8.58.2.21 dropped_fragment_count() . . . 1071
8.58.2.22 reassembled_sample_count() . . . 1071
8.58.2.23 sent_nack_fragment_count() . . . 1072
8.58.2.24 sent_nack_fragment_bytes() . . . 1072
8.58.2.25 out_of_range_rejected_sample_count() . . . 1072
8.59 dds::sub::qos::DataReaderQos Class Reference . . . 1072
8.59.1 Detailed Description . . . 1073
8.59.2 DataReaderQos policies . . . 1074
8.59.3 Constructor & Destructor Documentation . . . 1075
8.59.3.1 DataReaderQos() [1/2] . . . 1076
8.59.3.2 DataReaderQos() [2/2] . . . 1076
8.59.4 Member Function Documentation . . . 1076
8.59.4.1 operator=() . . . 1076
8.59.4.2 policy() [1/3] . . . 1077
8.59.4.3 policy() [2/3] . . . 1077
8.59.4.4 policy() [3/3] . . . 1078
8.59.4.5 operator<<() . . . 1078
8.59.4.6 operator>>() . . . 1078
8.59.5 Friends And Related Function Documentation . . . 1079
8.59.5.1 to_string() [1/3] . . . 1079
8.59.5.2 to_string() [2/3] . . . 1079
8.59.5.3 to_string() [3/3] . . . 1080
8.59.5.4 operator<<() . . . 1081
8.60 rti::core::policy::DataReaderResourceLimits Class Reference . . . 1081
8.60.1 Detailed Description . . . 1085
8.60.2 Constructor & Destructor Documentation . . . 1085
8.60.2.1 DataReaderResourceLimits() . . . 1085
8.60.3 Member Function Documentation . . . 1085
8.60.3.1 max_remote_writers() [1/2] . . . 1086

8.60.3.2 max_remote_writers() [2/2] . . . 1086
8.60.3.3 max_remote_writers_per_instance() [1/2] . . . 1086
8.60.3.4 max_remote_writers_per_instance() [2/2] . . . 1087
8.60.3.5 max_samples_per_remote_writer() [1/2] . . . 1087
8.60.3.6 max_samples_per_remote_writer() [2/2] . . . 1087
8.60.3.7 max_infos() [1/2] . . . 1088
8.60.3.8 max_infos() [2/2] . . . 1088
8.60.3.9 initial_remote_writers() [1/2] . . . 1088
8.60.3.10 initial_remote_writers() [2/2] . . . 1089
8.60.3.11 initial_remote_writers_per_instance() [1/2] . . . 1089
8.60.3.12 initial_remote_writers_per_instance() [2/2] . . . 1089
8.60.3.13 initial_infos() [1/2] . . . 1090
8.60.3.14 initial_infos() [2/2] . . . 1090
8.60.3.15 initial_outstanding_reads() [1/2] . . . 1090
8.60.3.16 initial_outstanding_reads() [2/2] . . . 1091
8.60.3.17 max_outstanding_reads() [1/2] . . . 1091
8.60.3.18 max_outstanding_reads() [2/2] . . . 1091
8.60.3.19 max_samples_per_read() [1/2] . . . 1091
8.60.3.20 max_samples_per_read() [2/2] . . . 1092
8.60.3.21 disable_fragmentation_support() [1/2] . . . 1092
8.60.3.22 disable_fragmentation_support() [2/2] . . . 1092
8.60.3.23 max_fragmented_samples() [1/2] . . . 1093
8.60.3.24 max_fragmented_samples() [2/2] . . . 1093
8.60.3.25 initial_fragmented_samples() [1/2] . . . 1093
8.60.3.26 initial_fragmented_samples() [2/2] . . . 1094
8.60.3.27 max_fragmented_samples_per_remote_writer() [1/2] . . . 1094
8.60.3.28 max_fragmented_samples_per_remote_writer() [2/2] . . . 1094
8.60.3.29 max_fragments_per_sample() [1/2] . . . 1095
8.60.3.30 max_fragments_per_sample() [2/2] . . . 1095
8.60.3.31 dynamically_allocate_fragmented_samples() [1/2] . . . 1095
8.60.3.32 dynamically_allocate_fragmented_samples() [2/2] . . . 1096
8.60.3.33 max_total_instances() [1/2] . . . 1096
8.60.3.34 max_total_instances() [2/2] . . . 1097
8.60.3.35 max_remote_virtual_writers() [1/2] . . . 1097
8.60.3.36 max_remote_virtual_writers() [2/2] . . . 1097
8.60.3.37 initial_remote_virtual_writers() [1/2] . . . 1098
8.60.3.38 initial_remote_virtual_writers() [2/2] . . . 1098
8.60.3.39 max_remote_virtual_writers_per_instance() [1/2] . . . 1098
8.60.3.40 max_remote_virtual_writers_per_instance() [2/2] . . . 1099

8.60.3.41 initial_remote_virtual_writers_per_instance() [1/2] . . . 1099
8.60.3.42 initial_remote_virtual_writers_per_instance() [2/2] . . . 1099
8.60.3.43 max_remote_writers_per_sample() [1/2] . . . 1100
8.60.3.44 max_remote_writers_per_sample() [2/2] . . . 1100
8.60.3.45 max_query_condition_filters() [1/2] . . . 1100
8.60.3.46 max_query_condition_filters() [2/2] . . . 1101
8.60.3.47 max_app_ack_response_length() [1/2] . . . 1101
8.60.3.48 max_app_ack_response_length() [2/2] . . . 1101
8.60.3.49 keep_minimum_state_for_instances() [1/2] . . . 1102
8.60.3.50 keep_minimum_state_for_instances() [2/2] . . . 1102
8.60.3.51 initial_topic_queries() [1/2] . . . 1102
8.60.3.52 initial_topic_queries() [2/2] . . . 1103
8.60.3.53 max_topic_queries() [1/2] . . . 1103
8.60.3.54 max_topic_queries() [2/2] . . . 1103
8.60.3.55 autopurge_remote_not_alive_writer_delay() [1/2] . . . 1104
8.60.3.56 autopurge_remote_not_alive_writer_delay() [2/2] . . . 1104
8.60.3.57 autopurge_remote_virtual_writer_delay() [1/2] . . . 1105
8.60.3.58 autopurge_remote_virtual_writer_delay() [2/2] . . . 1105
8.60.3.59 lazily_allocate_batch_resources() [1/2] . . . 1105
8.60.3.60 lazily_allocate_batch_resources() [2/2] . . . 1106
8.60.3.61 shmem_ref_transfer_mode_attached_segment_allocation() [1/2] . . . 1106
8.60.3.62 shmem_ref_transfer_mode_attached_segment_allocation() [2/2] . . . 1106
8.60.3.63 instance_replacement() [1/2] . . . 1107
8.60.3.64 instance_replacement() [2/2] . . . 1108
8.61 rti::core::DataReaderResourceLimitsInstanceReplacementSettings Class Reference . . . 1108
8.61.1 Detailed Description . . . 1109
8.61.2 Constructor & Destructor Documentation . . . 1109
8.61.2.1 DataReaderResourceLimitsInstanceReplacementSettings() [1/2] . . . 1109
8.61.2.2 DataReaderResourceLimitsInstanceReplacementSettings() [2/2] . . . 1109
8.61.3 Member Function Documentation . . . 1110
8.61.3.1 alive_instance_removal() [1/2] . . . 1110
8.61.3.2 alive_instance_removal() [2/2] . . . 1110
8.61.3.3 disposed_instance_removal() [1/2] . . . 1110
8.61.3.4 disposed_instance_removal() [2/2] . . . 1110
8.61.3.5 no_writers_instance_removal() [1/2] . . . 1111
8.61.3.6 no_writers_instance_removal() [2/2] . . . 1111
8.62 rti::sub::cond::DataReaderStatusConditionHandler< T > Class Template Reference . . . 1111
8.62.1 Detailed Description . . . 1111
8.62.2 Constructor & Destructor Documentation . . . 1112

8.62.2.1 DataReaderStatusConditionHandler() . . . 1112
8.63 dds::core::policy::DataRepresentation Class Reference . . . 1113
8.63.1 Detailed Description . . . 1114
8.63.2 Constructor & Destructor Documentation . . . 1115
8.63.2.1 DataRepresentation() [1/3] . . . 1115
8.63.2.2 DataRepresentation() [2/3] . . . 1115
8.63.2.3 DataRepresentation() [3/3] . . . 1115
8.63.3 Member Function Documentation . . . 1115
8.63.3.1 create_empty() . . . 1116
8.63.3.2 value() [1/3] . . . 1116
8.63.3.3 value() [2/3] . . . 1116
8.63.3.4 value() [3/3] . . . 1116
8.63.3.5 compression_settings() [1/3] . . . 1117
8.63.3.6 compression_settings() [2/3] . . . 1117
8.63.3.7 compression_settings() [3/3] . . . 1117
8.64 dds::sub::status::DataState Class Reference . . . 1118
8.64.1 Detailed Description . . . 1120
8.64.2 Constructor & Destructor Documentation . . . 1120
8.64.2.1 DataState() [1/5] . . . 1120
8.64.2.2 DataState() [2/5] . . . 1121
8.64.2.3 DataState() [3/5] . . . 1121
8.64.2.4 DataState() [4/5] . . . 1121
8.64.2.5 DataState() [5/5] . . . 1121
8.64.3 Member Function Documentation . . . 1121
8.64.3.1 operator==() . . . 1122
8.64.3.2 operator"!=() . . . 1122
8.64.3.3 operator<<() [1/3] . . . 1122
8.64.3.4 operator<<() [2/3] . . . 1122
8.64.3.5 operator<<() [3/3] . . . 1122
8.64.3.6 operator>>() [1/3] . . . 1123
8.64.3.7 operator>>() [2/3] . . . 1123
8.64.3.8 operator>>() [3/3] . . . 1123
8.64.3.9 sample_state() [1/2] . . . 1123
8.64.3.10 sample_state() [2/2] . . . 1123
8.64.3.11 instance_state() [1/2] . . . 1124
8.64.3.12 instance_state() [2/2] . . . 1124
8.64.3.13 view_state() [1/2] . . . 1124
8.64.3.14 view_state() [2/2] . . . 1124
8.64.3.15 any() . . . 1124

8.64.3.16 new_data() . . . 1125
8.64.3.17 any_data() . . . 1125
8.64.3.18 new_instance() . . . 1125
8.64.4 Friends And Related Function Documentation . . . 1125
8.64.4.1 swap . . . 1125
8.64.4.2 operator<<() . . . 1126
8.65 rti::sub::status::DataStateEx Class Reference . . . 1126
8.65.1 Detailed Description . . . 1127
8.65.2 Constructor & Destructor Documentation . . . 1128
8.65.2.1 DataStateEx() [1/4] . . . 1128
8.65.2.2 DataStateEx() [2/4] . . . 1128
8.65.2.3 DataStateEx() [3/4] . . . 1128
8.65.2.4 DataStateEx() [4/4] . . . 1128
8.65.3 Member Function Documentation . . . 1129
8.65.3.1 operator<<() . . . 1129
8.65.3.2 operator>>() . . . 1129
8.65.3.3 stream_kind() [1/2] . . . 1129
8.65.3.4 stream_kind() [2/2] . . . 1129
8.65.3.5 data_state() . . . 1130
8.65.3.6 operator==() . . . 1130
8.65.3.7 operator"!=() . . . 1130
8.65.3.8 sample_state() [1/2] . . . 1130
8.65.3.9 sample_state() [2/2] . . . 1130
8.65.3.10 view_state() [1/2] . . . 1131
8.65.3.11 view_state() [2/2] . . . 1131
8.65.3.12 instance_state() [1/2] . . . 1131
8.65.3.13 instance_state() [2/2] . . . 1131
8.65.3.14 any() . . . 1131
8.65.3.15 new_data() . . . 1132
8.65.3.16 any_data() . . . 1132
8.65.3.17 new_instance() . . . 1132
8.65.4 Friends And Related Function Documentation . . . 1132
8.65.4.1 swap . . . 1132
8.66 dds::core::policy::DataTag Class Reference . . . 1132
8.66.1 Detailed Description . . . 1133
8.66.2 Usage . . . 1134
8.66.3 Member Typedef Documentation . . . 1134
8.66.3.1 Entry . . . 1134
8.66.4 Constructor & Destructor Documentation . . . 1134

8.66.4.1 DataTag() [1/3] . . . 1134
8.66.4.2 DataTag() [2/3] . . . 1135
8.66.4.3 DataTag() [3/3] . . . 1135
8.66.5 Member Function Documentation . . . 1135
8.66.5.1 set() [1/3] . . . 1135
8.66.5.2 exists() . . . 1135
8.66.5.3 get() . . . 1136
8.66.5.4 try_get() . . . 1136
8.66.5.5 set() [2/3] . . . 1136
8.66.5.6 set() [3/3] . . . 1136
8.66.5.7 remove() . . . 1137
8.66.5.8 size() . . . 1137
8.66.5.9 get_all() . . . 1137
8.67 dds::pub::DataWriter< T > Class Template Reference . . . 1138
8.67.1 Detailed Description . . . 1143
8.67.2 Notes about DataWriter destruction . . . 1144
8.67.3 Constructor & Destructor Documentation . . . 1144
8.67.3.1 DataWriter() [1/6] . . . 1144
8.67.3.2 DataWriter() [2/6] . . . 1145
8.67.3.3 DataWriter() [3/6] . . . 1146
8.67.3.4 DataWriter() [4/6] . . . 1147
8.67.3.5 DataWriter() [5/6] . . . 1147
8.67.3.6 DataWriter() [6/6] . . . 1148
8.67.4 Member Function Documentation . . . 1148
8.67.4.1 write() [1/11] . . . 1149
8.67.4.2 write() [2/11] . . . 1149
8.67.4.3 write() [3/11] . . . 1150
8.67.4.4 write() [4/11] . . . 1153
8.67.4.5 write() [5/11] . . . 1154
8.67.4.6 write() [6/11] . . . 1155
8.67.4.7 write() [7/11] . . . 1155
8.67.4.8 write() [8/11] . . . 1156
8.67.4.9 write() [9/11] . . . 1156
8.67.4.10 write() [10/11] . . . 1157
8.67.4.11 operator<<() [1/4] . . . 1157
8.67.4.12 operator<<() [2/4] . . . 1158
8.67.4.13 operator<<() [3/4] . . . 1158
8.67.4.14 register_instance() [1/3] . . . 1158
8.67.4.15 register_instance() [2/3] . . . 1159

8.67.4.16 unregister_instance() [1/3] . . . 1160
8.67.4.17 unregister_instance() [2/3] . . . 1161
8.67.4.18 dispose_instance() [1/3] . . . 1162
8.67.4.19 dispose_instance() [2/3] . . . 1163
8.67.4.20 key_value() [1/2] . . . 1164
8.67.4.21 key_value() [2/2] . . . 1165
8.67.4.22 lookup_instance() . . . 1166
8.67.4.23 qos() [1/2] . . . 1166
8.67.4.24 qos() [2/2] . . . 1166
8.67.4.25 operator<<() [4/4] . . . 1167
8.67.4.26 operator>>() . . . 1167
8.67.4.27 topic() . . . 1168
8.67.4.28 publisher() . . . 1168
8.67.4.29 wait_for_acknowledgments() . . . 1168
8.67.4.30 listener() [1/2] . . . 1169
8.67.4.31 listener() [2/2] . . . 1169
8.67.4.32 set_listener() [1/2] . . . 1170
8.67.4.33 set_listener() [2/2] . . . 1170
8.67.4.34 get_listener() . . . 1171
8.67.4.35 liveliness_lost_status() . . . 1171
8.67.4.36 offered_deadline_missed_status() . . . 1171
8.67.4.37 offered_incompatible_qos_status() . . . 1172
8.67.4.38 publication_matched_status() . . . 1172
8.67.4.39 assert_liveliness() . . . 1172
8.67.4.40 unregister_instance() [3/3] . . . 1173
8.67.4.41 dispose_instance() [3/3] . . . 1173
8.67.4.42 is_sample_app_acknowledged() . . . 1174
8.67.4.43 wait_for_asynchronous_publishing() . . . 1174
8.67.4.44 reliable_writer_cache_changed_status() . . . 1175
8.67.4.45 reliable_reader_activity_changed_status() . . . 1176
8.67.4.46 datawriter_cache_status() . . . 1176
8.67.4.47 datawriter_protocol_status() . . . 1176
8.67.4.48 matched_subscription_datawriter_protocol_status() [1/2] . . . 1177
8.67.4.49 matched_subscription_datawriter_protocol_status() [2/2] . . . 1177
8.67.4.50 service_request_accepted_status() . . . 1178
8.67.4.51 flush() . . . 1179
8.67.4.52 write() [11/11] . . . 1179
8.67.4.53 write_noexcept() [1/5] . . . 1180
8.67.4.54 write_noexcept() [2/5] . . . 1181

8.67.4.55 write_noexcept() [3/5] . . . 1181
8.67.4.56 write_noexcept() [4/5] . . . 1182
8.67.4.57 write_noexcept() [5/5] . . . 1182
8.67.4.58 create_data() . . . 1182
8.67.4.59 delete_data() . . . 1183
8.67.4.60 get_loan() . . . 1183
8.67.4.61 discard_loan() . . . 1185
8.67.4.62 register_instance() [3/3] . . . 1185
8.67.5 Friends And Related Function Documentation . . . 1186
8.67.5.1 ignore() [1/2] . . . 1186
8.67.5.2 ignore() [2/2] . . . 1187
8.67.5.3 matched_subscriptions() [1/2] . . . 1187
8.67.5.4 matched_subscriptions() [2/2] . . . 1188
8.67.5.5 matched_subscription_data() [1/2] . . . 1189
8.67.5.6 matched_subscription_participant_data() . . . 1190
8.67.5.7 matched_subscriptions_locators() . . . 1191
8.67.5.8 is_matched_subscription_active() . . . 1191
8.67.5.9 matched_subscription_data() [2/2] . . . 1192
8.67.5.10 find() [1/2] . . . 1193
8.67.5.11 find() [2/2] . . . 1194
8.67.5.12 find_datawriters() [1/2] . . . 1195
8.67.5.13 find_datawriters() [2/2] . . . 1195
8.67.5.14 find_datawriter_by_topic_name() . . . 1196
8.67.5.15 find_datawriter_by_name() [1/2] . . . 1197
8.67.5.16 find_datawriter_by_name() [2/2] . . . 1198
8.68 rti::core::status::DataWriterCacheStatus Class Reference . . . 1199
8.68.1 Detailed Description . . . 1200
8.68.2 Member Function Documentation . . . 1200
8.68.2.1 sample_count() . . . 1200
8.68.2.2 sample_count_peak() . . . 1200
8.68.2.3 alive_instance_count() . . . 1201
8.68.2.4 alive_instance_count_peak() . . . 1201
8.68.2.5 disposed_instance_count() . . . 1201
8.68.2.6 disposed_instance_count_peak() . . . 1201
8.68.2.7 unregistered_instance_count() . . . 1201
8.68.2.8 unregistered_instance_count_peak() . . . 1202
8.69 dds::pub::DataWriterListener< T > Class Template Reference . . . 1202
8.69.1 Detailed Description . . . 1203
8.69.2 Member Function Documentation . . . 1203

8.69.2.1 on_offered_deadline_missed() . . . 1204
8.69.2.2 on_offered_incompatible_qos() . . . 1204
8.69.2.3 on_liveliness_lost() . . . 1205
8.69.2.4 on_publication_matched() . . . 1205
8.69.2.5 on_reliable_writer_cache_changed() . . . 1205
8.69.2.6 on_reliable_reader_activity_changed() . . . 1206
8.69.2.7 on_instance_replaced() . . . 1206
8.69.2.8 on_application_acknowledgment() . . . 1207
8.69.2.9 on_service_request_accepted() . . . 1209
8.69.2.10 on_sample_removed() . . . 1209
8.70 rti::core::policy::DataWriterProtocol Class Reference . . . 1210
8.70.1 Detailed Description . . . 1211
8.70.2 Constructor & Destructor Documentation . . . 1211
8.70.2.1 DataWriterProtocol() . . . 1212
8.70.3 Member Function Documentation . . . 1212
8.70.3.1 virtual_guid() [1/2] . . . 1212
8.70.3.2 virtual_guid() [2/2] . . . 1212
8.70.3.3 rtps_object_id() [1/2] . . . 1213
8.70.3.4 rtps_object_id() [2/2] . . . 1213
8.70.3.5 push_on_write() [1/2] . . . 1213
8.70.3.6 push_on_write() [2/2] . . . 1214
8.70.3.7 disable_positive_acks() [1/2] . . . 1214
8.70.3.8 disable_positive_acks() [2/2] . . . 1214
8.70.3.9 disable_inline_keyhash() [1/2] . . . 1215
8.70.3.10 disable_inline_keyhash() [2/2] . . . 1215
8.70.3.11 serialize_key_with_dispose() [1/2] . . . 1216
8.70.3.12 serialize_key_with_dispose() [2/2] . . . 1216
8.70.3.13 propagate_app_ack_with_no_response() [1/2] . . . 1217
8.70.3.14 propagate_app_ack_with_no_response() [2/2] . . . 1217
8.70.3.15 rtps_reliable_writer() [1/3] . . . 1217
8.70.3.16 rtps_reliable_writer() [2/3] . . . 1218
8.70.3.17 rtps_reliable_writer() [3/3] . . . 1218
8.70.3.18 initial_virtual_sequence_number() . . . 1218
8.71 rti::core::status::DataWriterProtocolStatus Class Reference . . . 1219
8.71.1 Detailed Description . . . 1220
8.71.2 Member Function Documentation . . . 1220
8.71.2.1 pushed_sample_count() . . . 1220
8.71.2.2 pushed_sample_bytes() . . . 1221
8.71.2.3 filtered_sample_count() . . . 1221

8.71.2.4 filtered_sample_bytes() . . . 1221
8.71.2.5 sent_heartbeat_count() . . . 1221
8.71.2.6 sent_heartbeat_bytes() . . . 1221
8.71.2.7 pulled_sample_count() . . . 1222
8.71.2.8 pulled_sample_bytes() . . . 1222
8.71.2.9 received_ack_count() . . . 1222
8.71.2.10 received_ack_bytes() . . . 1222
8.71.2.11 received_nack_count() . . . 1222
8.71.2.12 received_nack_bytes() . . . 1223
8.71.2.13 sent_gap_count() . . . 1223
8.71.2.14 sent_gap_bytes() . . . 1223
8.71.2.15 rejected_sample_count() . . . 1223
8.71.2.16 send_window_size() . . . 1223
8.71.2.17 first_available_sample_sequence_number() . . . 1224
8.71.2.18 last_available_sample_sequence_number() . . . 1224
8.71.2.19 first_unacknowledged_sample_sequence_number() . . . 1224
8.71.2.20 first_available_sample_virtual_sequence_number() . . . 1224
8.71.2.21 last_available_sample_virtual_sequence_number() . . . 1224
8.71.2.22 first_unacknowledged_sample_virtual_sequence_number() . . . 1225
8.71.2.23 first_unacknowledged_sample_subscription_handle() . . . 1225
8.71.2.24 first_unelapsed_keep_duration_sample_sequence_number() . . . 1225
8.71.2.25 pushed_fragment_count() . . . 1225
8.71.2.26 pushed_fragment_bytes() . . . 1225
8.71.2.27 pulled_fragment_count() . . . 1226
8.71.2.28 pulled_fragment_bytes() . . . 1226
8.71.2.29 received_nack_fragment_count() . . . 1226
8.71.2.30 received_nack_fragment_bytes() . . . 1226
8.72 dds::pub::qos::DataWriterQos Class Reference . . . 1226
8.72.1 Detailed Description . . . 1227
8.72.2 DataWriterQos policies . . . 1228
8.72.3 Constructor & Destructor Documentation . . . 1229
8.72.3.1 DataWriterQos() [1/2] . . . 1230
8.72.3.2 DataWriterQos() [2/2] . . . 1230
8.72.4 Member Function Documentation . . . 1230
8.72.4.1 operator=() . . . 1230
8.72.4.2 policy() [1/3] . . . 1231
8.72.4.3 policy() [2/3] . . . 1231
8.72.4.4 policy() [3/3] . . . 1232
8.72.4.5 operator<<() . . . 1232

8.72.4.6 operator>>() 1232
8.72.5 Friends And Related Function Documentation 1233
8.72.5.1 to_string() [1/3] 1233
8.72.5.2 to_string() [2/3] 1233
8.72.5.3 to_string() [3/3] 1234
8.72.5.4 operator<<() 1235
8.73 rti::core::policy::DataWriterResourceLimits Class Reference 1235
8.73.1 Detailed Description 1237
8.73.2 Constructor & Destructor Documentation 1237
8.73.2.1 DataWriterResourceLimits() 1237
8.73.3 Member Function Documentation 1238
8.73.3.1 initial_concurrent_blocking_threads() [1/2] 1238
8.73.3.2 initial_concurrent_blocking_threads() [2/2] 1238
8.73.3.3 max_concurrent_blocking_threads() [1/2] 1238
8.73.3.4 max_concurrent_blocking_threads() [2/2] 1239
8.73.3.5 max_remote_reader_filters() [1/2] 1239
8.73.3.6 max_remote_reader_filters() [2/2] 1239
8.73.3.7 initial_batches() [1/2] 1240
8.73.3.8 initial_batches() [2/2] 1240
8.73.3.9 max_batches() [1/2] 1240
8.73.3.10 max_batches() [2/2] 1241
8.73.3.11 cookie_max_length() [1/2] 1241
8.73.3.12 cookie_max_length() [2/2] 1241
8.73.3.13 instance_replacement() [1/2] 1241
8.73.3.14 instance_replacement() [2/2] 1242
8.73.3.15 replace_empty_instances() [1/2] 1242
8.73.3.16 replace_empty_instances() [2/2] 1242
8.73.3.17 autoregister_instances() [1/2] 1243
8.73.3.18 autoregister_instances() [2/2] 1243
8.73.3.19 initial_virtual_writers() [1/2] 1243
8.73.3.20 initial_virtual_writers() [2/2] 1244
8.73.3.21 max_virtual_writers() [1/2] 1244
8.73.3.22 max_virtual_writers() [2/2] 1244
8.73.3.23 initial_remote_readers() [1/2] 1244
8.73.3.24 max_remote_readers() [1/2] 1245
8.73.3.25 initial_remote_readers() [2/2] 1245
8.73.3.26 max_remote_readers() [2/2] 1245
8.73.3.27 max_app_ack_remote_readers() [1/2] 1245
8.73.3.28 max_app_ack_remote_readers() [2/2] 1246

8.73.3.29 initial_active_topic_queries() [1/2] . . . 1246
8.73.3.30 initial_active_topic_queries() [2/2] . . . 1246
8.73.3.31 max_active_topic_queries() [1/2] . . . 1247
8.73.3.32 max_active_topic_queries() [2/2] . . . 1247
8.73.3.33 writer_loaned_sample_allocation() [1/2] . . . 1248
8.73.3.34 writer_loaned_sample_allocation() [2/2] . . . 1248
8.73.3.35 initialize_writer_loaned_sample() [1/2] . . . 1249
8.73.3.36 initialize_writer_loaned_sample() [2/2] . . . 1249
8.74 rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_def Struct Reference . . . 1249
8.74.1 Detailed Description . . . 1250
8.74.2 Member Enumeration Documentation . . . 1250
8.74.2.1 type . . . 1250
8.75 rti::core::DataWriterShmemRefTransferModeSettings Class Reference . . . 1251
8.75.1 Detailed Description . . . 1252
8.75.2 Constructor & Destructor Documentation . . . 1252
8.75.2.1 DataWriterShmemRefTransferModeSettings() . . . 1252
8.75.3 Member Function Documentation . . . 1252
8.75.3.1 enable_data_consistency_check() [1/2] . . . 1253
8.75.3.2 enable_data_consistency_check() [2/2] . . . 1253
8.76 rti::core::policy::DataWriterTransferMode Class Reference . . . 1253
8.76.1 Detailed Description . . . 1254
8.76.2 Constructor & Destructor Documentation . . . 1254
8.76.2.1 DataWriterTransferMode() . . . 1254
8.76.3 Member Function Documentation . . . 1254
8.76.3.1 ShmemRefSettings() . . . 1254
8.76.3.2 shmem_ref_settings() [1/3] . . . 1255
8.76.3.3 shmem_ref_settings() [2/3] . . . 1255
8.76.3.4 shmem_ref_settings() [3/3] . . . 1255
8.77 dds::core::policy::Deadline Class Reference . . . 1255
8.77.1 Detailed Description . . . 1256
8.77.2 Usage . . . 1256
8.77.3 Compatibility . . . 1257
8.77.4 Consistency . . . 1257
8.77.5 Constructor & Destructor Documentation . . . 1257
8.77.5.1 Deadline() [1/2] . . . 1257
8.77.5.2 Deadline() [2/2] . . . 1257
8.77.6 Member Function Documentation . . . 1258
8.77.6.1 period() [1/2] . . . 1258
8.77.6.2 period() [2/2] . . . 1258

8.78 dds::core::policy::DestinationOrder Class Reference . . . 1258
8.78.1 Detailed Description . . . 1259
8.78.2 Usage . . . 1259
8.78.3 Compatibility . . . 1260
8.78.4 Constructor & Destructor Documentation . . . 1260
8.78.4.1 DestinationOrder() [1/2] . . . 1261
8.78.4.2 DestinationOrder() [2/2] . . . 1261
8.78.5 Member Function Documentation . . . 1261
8.78.5.1 kind() [1/2] . . . 1261
8.78.5.2 kind() [2/2] . . . 1261
8.78.5.3 SourceTimestamp() . . . 1262
8.78.5.4 ReceptionTimestamp() . . . 1262
8.78.5.5 scope() [1/2] . . . 1262
8.78.5.6 scope() [2/2] . . . 1262
8.78.5.7 source_timestamp_tolerance() [1/2] . . . 1263
8.78.5.8 source_timestamp_tolerance() [2/2] . . . 1263
8.79 dds::core::policy::DestinationOrderKind_def Struct Reference . . . 1263
8.79.1 Detailed Description . . . 1264
8.79.2 Member Enumeration Documentation . . . 1264
8.79.2.1 type . . . 1264
8.80 rti::core::policy::DestinationOrderScopeKind_def Struct Reference . . . 1264
8.80.1 Detailed Description . . . 1265
8.80.2 Member Enumeration Documentation . . . 1265
8.80.2.1 type . . . 1265
8.81 rti::test::DisableMonitoringLibraryGuard Struct Reference . . . 1265
8.81.1 Detailed Description . . . 1265
8.82 rti::core::policy::Discovery Class Reference . . . 1265
8.82.1 Detailed Description . . . 1266
8.82.2 Usage . . . 1267
8.82.3 Constructor & Destructor Documentation . . . 1267
8.82.3.1 Discovery() . . . 1267
8.82.4 Member Function Documentation . . . 1267
8.82.4.1 enabled_transports() [1/2] . . . 1267
8.82.4.2 enabled_transports() [2/2] . . . 1268
8.82.4.3 initial_peers() [1/2] . . . 1268
8.82.4.4 initial_peers() [2/2] . . . 1268
8.82.4.5 multicast_receive_addresses() [1/2] . . . 1269
8.82.4.6 multicast_receive_addresses() [2/2] . . . 1269
8.82.4.7 metatraffic_transport_priority() [1/2] . . . 1270

8.82.4.8 metatraffic_transport_priority() [2/2] . . . 1270
8.82.4.9 accept_unknown_peers() [1/2] . . . 1270
8.82.4.10 accept_unknown_peers() [2/2] . . . 1271
8.82.4.11 enable_endpoint_discovery() [1/2] . . . 1271
8.82.4.12 enable_endpoint_discovery() [2/2] . . . 1271
8.83 rti::core::policy::DiscoveryConfig Class Reference . . . 1271
8.83.1 Detailed Description . . . 1278
8.83.2 Constructor & Destructor Documentation . . . 1279
8.83.2.1 DiscoveryConfig() . . . 1279
8.83.3 Member Function Documentation . . . 1279
8.83.3.1 participant_liveliness_lease_duration() [1/2] . . . 1279
8.83.3.2 participant_liveliness_lease_duration() [2/2] . . . 1279
8.83.3.3 participant_liveliness_assert_period() [1/2] . . . 1280
8.83.3.4 participant_liveliness_assert_period() [2/2] . . . 1280
8.83.3.5 participant_announcement_period() [1/2] . . . 1280
8.83.3.6 participant_announcement_period() [2/2] . . . 1281
8.83.3.7 remote_participant_purge_kind() [1/2] . . . 1281
8.83.3.8 remote_participant_purge_kind() [2/2] . . . 1281
8.83.3.9 max_liveliness_loss_detection_period() [1/2] . . . 1282
8.83.3.10 max_liveliness_loss_detection_period() [2/2] . . . 1282
8.83.3.11 initial_participant_announcements() [1/2] . . . 1282
8.83.3.12 initial_participant_announcements() [2/2] . . . 1283
8.83.3.13 new_remote_participant_announcements() [1/2] . . . 1283
8.83.3.14 new_remote_participant_announcements() [2/2] . . . 1283
8.83.3.15 min_initial_participant_announcement_period() [1/2] . . . 1284
8.83.3.16 min_initial_participant_announcement_period() [2/2] . . . 1284
8.83.3.17 max_initial_participant_announcement_period() [1/2] . . . 1285
8.83.3.18 max_initial_participant_announcement_period() [2/2] . . . 1285
8.83.3.19 participant_reader_resource_limits() [1/3] . . . 1285
8.83.3.20 participant_reader_resource_limits() [2/3] . . . 1286
8.83.3.21 participant_reader_resource_limits() [3/3] . . . 1286
8.83.3.22 publication_reader() [1/3] . . . 1286
8.83.3.23 publication_reader() [2/3] . . . 1287
8.83.3.24 publication_reader() [3/3] . . . 1287
8.83.3.25 publication_reader_resource_limits() [1/3] . . . 1287
8.83.3.26 publication_reader_resource_limits() [2/3] . . . 1287
8.83.3.27 publication_reader_resource_limits() [3/3] . . . 1288
8.83.3.28 subscription_reader() [1/3] . . . 1288
8.83.3.29 subscription_reader() [2/3] . . . 1288

8.83.3.30 subscription_reader() [3/3] 1289
8.83.3.31 subscription_reader_resource_limits() [1/3] 1289
8.83.3.32 subscription_reader_resource_limits() [2/3] 1289
8.83.3.33 subscription_reader_resource_limits() [3/3] 1289
8.83.3.34 publication_writer() [1/3] 1290
8.83.3.35 publication_writer() [2/3] 1290
8.83.3.36 publication_writer() [3/3] 1291
8.83.3.37 publication_writer_data_lifecycle() [1/3] 1291
8.83.3.38 publication_writer_data_lifecycle() [2/3] 1291
8.83.3.39 publication_writer_data_lifecycle() [3/3] 1291
8.83.3.40 subscription_writer() [1/3] 1292
8.83.3.41 subscription_writer() [2/3] 1292
8.83.3.42 subscription_writer() [3/3] 1293
8.83.3.43 subscription_writer_data_lifecycle() [1/3] 1293
8.83.3.44 subscription_writer_data_lifecycle() [2/3] 1293
8.83.3.45 subscription_writer_data_lifecycle() [3/3] 1293
8.83.3.46 builtin_discovery_plugins() [1/2] 1294
8.83.3.47 builtin_discovery_plugins() [2/2] 1294
8.83.3.48 enabled_builtin_channels() [1/2] 1294
8.83.3.49 enabled_builtin_channels() [2/2] 1295
8.83.3.50 participant_message_reader_reliability_kind() [1/2] 1295
8.83.3.51 participant_message_reader_reliability_kind() [2/2] 1295
8.83.3.52 participant_message_reader() [1/3] 1296
8.83.3.53 participant_message_reader() [2/3] 1296
8.83.3.54 participant_message_reader() [3/3] 1296
8.83.3.55 participant_message_writer() [1/3] 1297
8.83.3.56 participant_message_writer() [2/3] 1297
8.83.3.57 participant_message_writer() [3/3] 1298
8.83.3.58 publication_writer_publish_mode() [1/3] 1298
8.83.3.59 publication_writer_publish_mode() [2/3] 1298
8.83.3.60 publication_writer_publish_mode() [3/3] 1298
8.83.3.61 subscription_writer_publish_mode() [1/3] 1299
8.83.3.62 subscription_writer_publish_mode() [2/3] 1299
8.83.3.63 subscription_writer_publish_mode() [3/3] 1299
8.83.3.64 asynchronous_publisher() [1/3] 1299
8.83.3.65 asynchronous_publisher() [2/3] 1300
8.83.3.66 asynchronous_publisher() [3/3] 1300
8.83.3.67 default_domain_announcement_period() [1/2] 1300
8.83.3.68 default_domain_announcement_period() [2/2] 1301

8.83.3.69 ignore_default_domain_announcements() [1/2] . . . 1301
8.83.3.70 ignore_default_domain_announcements() [2/2] . . . 1301
8.83.3.71 service_request_writer() [1/3] . . . 1302
8.83.3.72 service_request_writer() [2/3] . . . 1302
8.83.3.73 service_request_writer() [3/3] . . . 1303
8.83.3.74 service_request_writer_data_lifecycle() [1/3] . . . 1303
8.83.3.75 service_request_writer_data_lifecycle() [2/3] . . . 1303
8.83.3.76 service_request_writer_data_lifecycle() [3/3] . . . 1303
8.83.3.77 service_request_writer_publish_mode() [1/3] . . . 1304
8.83.3.78 service_request_writer_publish_mode() [2/3] . . . 1304
8.83.3.79 service_request_writer_publish_mode() [3/3] . . . 1304
8.83.3.80 service_request_reader() [1/3] . . . 1304
8.83.3.81 service_request_reader() [2/3] . . . 1305
8.83.3.82 service_request_reader() [3/3] . . . 1305
8.83.3.83 locator_reachability_assert_period() [1/2] . . . 1305
8.83.3.84 locator_reachability_assert_period() [2/2] . . . 1306
8.83.3.85 locator_reachability_lease_duration() [1/2] . . . 1306
8.83.3.86 locator_reachability_lease_duration() [2/2] . . . 1306
8.83.3.87 locator_reachability_change_detection_period() [1/2] . . . 1307
8.83.3.88 locator_reachability_change_detection_period() [2/2] . . . 1307
8.83.3.89 secure_volatile_writer() [1/3] . . . 1308
8.83.3.90 secure_volatile_writer() [2/3] . . . 1308
8.83.3.91 secure_volatile_writer() [3/3] . . . 1309
8.83.3.92 secure_volatile_writer_publish_mode() [1/3] . . . 1309
8.83.3.93 secure_volatile_writer_publish_mode() [2/3] . . . 1309
8.83.3.94 secure_volatile_writer_publish_mode() [3/3] . . . 1309
8.83.3.95 secure_volatile_reader() [1/3] . . . 1310
8.83.3.96 secure_volatile_reader() [2/3] . . . 1310
8.83.3.97 secure_volatile_reader() [3/3] . . . 1310
8.83.3.98 endpoint_type_object_lb_serialization_threshold() [1/2] . . . 1311
8.83.3.99 endpoint_type_object_lb_serialization_threshold() [2/2] . . . 1311
8.83.3.100 dns_tracker_polling_period() [1/2] . . . 1312
8.83.3.101 dns_tracker_polling_period() [2/2] . . . 1312
8.83.3.102 participant_configuration_reader() [1/3] . . . 1313
8.83.3.103 participant_configuration_reader() [2/3] . . . 1313
8.83.3.104 participant_configuration_reader() [3/3] . . . 1313
8.83.3.105 participant_configuration_reader_resource_limits() [1/3] . . . 1314
8.83.3.106 participant_configuration_reader_resource_limits() [2/3] . . . 1314
8.83.3.107 participant_configuration_reader_resource_limits() [3/3] . . . 1314

8.83.3.108 participant_configuration_writer() [1/3] . . . 1315
8.83.3.109 participant_configuration_writer() [2/3] . . . 1315
8.83.3.110 participant_configuration_writer() [3/3] . . . 1316
8.83.3.111 participant_configuration_writer_data_lifecycle() [1/3] . . . 1316
8.83.3.112 participant_configuration_writer_data_lifecycle() [2/3] . . . 1316
8.83.3.113 participant_configuration_writer_data_lifecycle() [3/3] . . . 1316
8.83.3.114 participant_configuration_writer_publish_mode() [1/3] . . . 1317
8.83.3.115 participant_configuration_writer_publish_mode() [2/3] . . . 1317
8.83.3.116 participant_configuration_writer_publish_mode() [3/3] . . . 1317
8.83.3.117 request_types_filter() [1/3] . . . 1318
8.83.3.118 request_types_filter() [2/3] . . . 1318
8.83.3.119 request_types_filter() [3/3] . . . 1318
8.84 rti::core::policy::DiscoveryConfigBuiltinChannelKindMask Class Reference . . . 1319
8.84.1 Detailed Description . . . 1319
8.84.2 Member Typedef Documentation . . . 1320
8.84.2.1 MaskType . . . 1320
8.84.3 Constructor & Destructor Documentation . . . 1320
8.84.3.1 DiscoveryConfigBuiltinChannelKindMask() [1/3] . . . 1320
8.84.3.2 DiscoveryConfigBuiltinChannelKindMask() [2/3] . . . 1320
8.84.3.3 DiscoveryConfigBuiltinChannelKindMask() [3/3] . . . 1320
8.84.4 Member Function Documentation . . . 1321
8.84.4.1 service_request() . . . 1321
8.84.4.2 type_lookup_service() . . . 1321
8.84.4.3 type_lookup_service_request() . . . 1321
8.84.4.4 type_lookup_service_reply() . . . 1322
8.85 rti::core::policy::DiscoveryConfigBuiltinPluginKindMask Class Reference . . . 1322
8.85.1 Detailed Description . . . 1323
8.85.2 Member Typedef Documentation . . . 1323
8.85.2.1 MaskType . . . 1323
8.85.3 Constructor & Destructor Documentation . . . 1323
8.85.3.1 DiscoveryConfigBuiltinPluginKindMask() [1/3] . . . 1323
8.85.3.2 DiscoveryConfigBuiltinPluginKindMask() [2/3] . . . 1323
8.85.3.3 DiscoveryConfigBuiltinPluginKindMask() [3/3] . . . 1323
8.85.4 Member Function Documentation . . . 1324
8.85.4.1 none() . . . 1324
8.85.4.2 SPDP() . . . 1324
8.85.4.3 SEDP() . . . 1324
8.85.4.4 SPDP2() . . . 1325
8.85.4.5 DPSE() . . . 1325

8.85.4.6 SDP() . . . 1325
8.85.4.7 SDP2() . . . 1326
8.86 dds::domain::DomainParticipant Class Reference . . . 1326
8.86.1 Detailed Description . . . 1330
8.86.2 Member Typedef Documentation . . . 1331
8.86.2.1 Listener . . . 1331
8.86.3 Constructor & Destructor Documentation . . . 1332
8.86.3.1 DomainParticipant() [1/3] . . . 1332
8.86.3.2 DomainParticipant() [2/3] . . . 1332
8.86.3.3 DomainParticipant() [3/3] . . . 1333
8.86.4 Member Function Documentation . . . 1334
8.86.4.1 listener() [1/2] . . . 1334
8.86.4.2 listener() [2/2] . . . 1335
8.86.4.3 set_listener() [1/2] . . . 1335
8.86.4.4 set_listener() [2/2] . . . 1336
8.86.4.5 get_listener() . . . 1336
8.86.4.6 qos() [1/2] . . . 1337
8.86.4.7 qos() [2/2] . . . 1337
8.86.4.8 operator<<() . . . 1337
8.86.4.9 operator>>() . . . 1338
8.86.4.10 domain_id() . . . 1338
8.86.4.11 assert_liveliness() . . . 1338
8.86.4.12 property() . . . 1339
8.86.4.13 contains_entity() . . . 1339
8.86.4.14 current_time() . . . 1340
8.86.4.15 participant_factory_qos() [1/2] . . . 1340
8.86.4.16 participant_factory_qos() [2/2] . . . 1341
8.86.4.17 finalize_participant_factory() . . . 1341
8.86.4.18 default_participant_qos() [1/2] . . . 1341
8.86.4.19 default_participant_qos() [2/2] . . . 1341
8.86.4.20 default_publisher_qos() [1/2] . . . 1342
8.86.4.21 default_publisher_qos() [2/2] . . . 1342
8.86.4.22 default_subscriber_qos() [1/2] . . . 1343
8.86.4.23 default_subscriber_qos() [2/2] . . . 1344
8.86.4.24 default_topic_qos() [1/2] . . . 1344
8.86.4.25 default_topic_qos() [2/2] . . . 1345
8.86.4.26 close_contained_entities() . . . 1346
8.86.4.27 default_datawriter_qos() [1/2] . . . 1346
8.86.4.28 default_datawriter_qos() [2/2] . . . 1347

8.86.4.29 default_datareader_qos() [1/2] . . . 1348
8.86.4.30 default_datareader_qos() [2/2] . . . 1349
8.86.4.31 register_contentfilter() . . . 1349
8.86.4.32 unregister_contentfilter() . . . 1350
8.86.4.33 unregister_type() . . . 1351
8.86.4.34 is_type_registered() . . . 1352
8.86.4.35 add_peer() . . . 1352
8.86.4.36 remove_peer() . . . 1353
8.86.4.37 dns_tracker_polling_period() [1/2] . . . 1355
8.86.4.38 dns_tracker_polling_period() [2/2] . . . 1355
8.86.4.39 resume_endpoint_discovery() . . . 1356
8.86.4.40 delete_durable_subscription() . . . 1357
8.86.4.41 register_durable_subscription() . . . 1358
8.86.4.42 participant_protocol_status() . . . 1359
8.86.4.43 time_until_license_expiration() . . . 1359
8.86.5 Friends And Related Function Documentation . . . 1359
8.86.5.1 ignore() [1/2] . . . 1360
8.86.5.2 ignore() [2/2] . . . 1360
8.86.5.3 discovered_participants() [1/2] . . . 1361
8.86.5.4 discovered_participants() [2/2] . . . 1362
8.86.5.5 discovered_participant_data() . . . 1362
8.86.5.6 find() . . . 1363
8.86.5.7 banish_ignored_participants() . . . 1364
8.86.5.8 discovered_participant_subject_name() . . . 1365
8.86.5.9 discovered_participants_from_subject_name() . . . 1366
8.86.5.10 find_participant_by_name() . . . 1367
8.86.5.11 find_participants() [1/2] . . . 1367
8.86.5.12 find_participants() [2/2] . . . 1368
8.86.5.13 find_type() . . . 1369
8.86.5.14 register_type() [1/2] . . . 1370
8.86.5.15 register_type() [2/2] . . . 1370
8.87 rti::domain::DomainParticipantConfigParams Class Reference . . . 1371
8.87.1 Detailed Description . . . 1372
8.87.2 Constructor & Destructor Documentation . . . 1373
8.87.2.1 DomainParticipantConfigParams() . . . 1373
8.87.3 Member Function Documentation . . . 1373
8.87.3.1 domain_id() [1/2] . . . 1373
8.87.3.2 domain_id() [2/2] . . . 1374
8.87.3.3 participant_name() [1/2] . . . 1374

8.87.3.4 participant_name() [2/2] . . . 1374
8.87.3.5 participant_qos_library_name() [1/2] . . . 1374
8.87.3.6 participant_qos_library_name() [2/2] . . . 1375
8.87.3.7 participant_qos_profile_name() [1/2] . . . 1375
8.87.3.8 participant_qos_profile_name() [2/2] . . . 1375
8.87.3.9 domain_entity_qos_library_name() [1/2] . . . 1376
8.87.3.10 domain_entity_qos_library_name() [2/2] . . . 1376
8.87.3.11 domain_entity_qos_profile_name() [1/2] . . . 1376
8.87.3.12 domain_entity_qos_profile_name() [2/2] . . . 1377
8.87.4 Member Data Documentation . . . 1377
8.87.4.1 ENTITY_NAME_USE_XML_CONFIG . . . 1377
8.87.4.2 QOS_ELEMENT_NAME_USE_XML_CONFIG . . . 1377
8.87.4.3 DOMAIN_ID_USE_XML_CONFIG . . . 1377
8.88 dds::domain::qos::DomainParticipantFactoryQos Class Reference . . . 1378
8.88.1 Detailed Description . . . 1378
8.88.2 Member Function Documentation . . . 1379
8.88.2.1 policy() [1/2] . . . 1379
8.88.2.2 policy() [2/2] . . . 1379
8.88.3 Friends And Related Function Documentation . . . 1379
8.88.3.1 to_string() [1/3] . . . 1380
8.88.3.2 to_string() [2/3] . . . 1380
8.88.3.3 to_string() [3/3] . . . 1381
8.88.3.4 operator<<() . . . 1382
8.89 dds::domain::DomainParticipantListener Class Reference . . . 1382
8.89.1 Detailed Description . . . 1382
8.89.2 Member Function Documentation . . . 1382
8.89.2.1 on_invalid_local_status_advance_notice() . . . 1383
8.90 rti::core::status::DomainParticipantProtocolStatus Class Reference . . . 1383
8.90.1 Detailed Description . . . 1383
8.90.2 Member Function Documentation . . . 1383
8.90.2.1 corrupted_rtps_message_count() . . . 1383
8.90.2.2 corrupted_rtps_message_count_change() . . . 1384
8.90.2.3 last_corrupted_message_timestamp() . . . 1384
8.91 dds::domain::qos::DomainParticipantQos Class Reference . . . 1384
8.91.1 Detailed Description . . . 1385
8.91.2 DomainParticipantQos Policies . . . 1385
8.91.3 Constructor & Destructor Documentation . . . 1386
8.91.3.1 DomainParticipantQos() . . . 1386
8.91.4 Member Function Documentation . . . 1387

8.91.4.1 policy() [1/3] . . . 1387
8.91.4.2 policy() [2/3] . . . 1387
8.91.4.3 policy() [3/3] . . . 1387
8.91.4.4 operator<<() . . . 1388
8.91.4.5 operator>>() . . . 1388
8.91.5 Friends And Related Function Documentation . . . 1388
8.91.5.1 to_string() [1/3] . . . 1389
8.91.5.2 to_string() [2/3] . . . 1389
8.91.5.3 to_string() [3/3] . . . 1390
8.91.5.4 operator<<() . . . 1390
8.92 rti::core::policy::DomainParticipantResourceLimits Class Reference . . . 1391
8.92.1 Detailed Description . . . 1397
8.92.2 Member Function Documentation . . . 1398
8.92.2.1 local_writer_allocation() [1/2] . . . 1398
8.92.2.2 local_writer_allocation() [2/2] . . . 1398
8.92.2.3 local_reader_allocation() [1/2] . . . 1398
8.92.2.4 local_reader_allocation() [2/2] . . . 1399
8.92.2.5 local_publisher_allocation() [1/2] . . . 1399
8.92.2.6 local_publisher_allocation() [2/2] . . . 1399
8.92.2.7 local_subscriber_allocation() . . . 1399
8.92.2.8 local_topic_allocation() [1/2] . . . 1400
8.92.2.9 local_topic_allocation() [2/2] . . . 1400
8.92.2.10 remote_writer_allocation() [1/2] . . . 1400
8.92.2.11 remote_writer_allocation() [2/2] . . . 1401
8.92.2.12 remote_reader_allocation() [1/2] . . . 1401
8.92.2.13 remote_reader_allocation() [2/2] . . . 1401
8.92.2.14 remote_participant_allocation() [1/2] . . . 1401
8.92.2.15 remote_participant_allocation() [2/2] . . . 1402
8.92.2.16 matching_writer_reader_pair_allocation() [1/2] . . . 1402
8.92.2.17 matching_writer_reader_pair_allocation() [2/2] . . . 1402
8.92.2.18 matching_reader_writer_pair_allocation() [1/2] . . . 1402
8.92.2.19 matching_reader_writer_pair_allocation() [2/2] . . . 1403
8.92.2.20 ignored_entity_allocation() [1/2] . . . 1403
8.92.2.21 ignored_entity_allocation() [2/2] . . . 1403
8.92.2.22 content_filtered_topic_allocation() [1/2] . . . 1403
8.92.2.23 content_filtered_topic_allocation() [2/2] . . . 1404
8.92.2.24 content_filter_allocation() [1/2] . . . 1404
8.92.2.25 content_filter_allocation() [2/2] . . . 1404
8.92.2.26 read_condition_allocation() [1/2] . . . 1404

8.92.2.27 read_condition_allocation() [2/2] . . . 1405
8.92.2.28 query_condition_allocation() [1/2] . . . 1405
8.92.2.29 query_condition_allocation() [2/2] . . . 1405
8.92.2.30 outstanding_asynchronous_sample_allocation() [1/2] . . . 1405
8.92.2.31 outstanding_asynchronous_sample_allocation() [2/2] . . . 1406
8.92.2.32 flow_controller_allocation() [1/2] . . . 1406
8.92.2.33 flow_controller_allocation() [2/2] . . . 1406
8.92.2.34 local_writer_hash_buckets() [1/2] . . . 1406
8.92.2.35 local_writer_hash_buckets() [2/2] . . . 1407
8.92.2.36 local_reader_hash_buckets() [1/2] . . . 1407
8.92.2.37 local_reader_hash_buckets() [2/2] . . . 1407
8.92.2.38 local_publisher_hash_buckets() [1/2] . . . 1407
8.92.2.39 local_publisher_hash_buckets() [2/2] . . . 1408
8.92.2.40 local_subscriber_hash_buckets() [1/2] . . . 1408
8.92.2.41 local_subscriber_hash_buckets() [2/2] . . . 1408
8.92.2.42 local_topic_hash_buckets() [1/2] . . . 1408
8.92.2.43 local_topic_hash_buckets() [2/2] . . . 1409
8.92.2.44 remote_writer_hash_buckets() [1/2] . . . 1409
8.92.2.45 remote_writer_hash_buckets() [2/2] . . . 1409
8.92.2.46 remote_reader_hash_buckets() [1/2] . . . 1409
8.92.2.47 remote_reader_hash_buckets() [2/2] . . . 1410
8.92.2.48 remote_participant_hash_buckets() [1/2] . . . 1410
8.92.2.49 remote_participant_hash_buckets() [2/2] . . . 1410
8.92.2.50 matching_writer_reader_pair_hash_buckets() [1/2] . . . 1410
8.92.2.51 matching_writer_reader_pair_hash_buckets() [2/2] . . . 1411
8.92.2.52 matching_reader_writer_pair_hash_buckets() [1/2] . . . 1411
8.92.2.53 matching_reader_writer_pair_hash_buckets() [2/2] . . . 1411
8.92.2.54 ignored_entity_hash_buckets() [1/2] . . . 1411
8.92.2.55 ignored_entity_hash_buckets() [2/2] . . . 1412
8.92.2.56 content_filtered_topic_hash_buckets() [1/2] . . . 1412
8.92.2.57 content_filtered_topic_hash_buckets() [2/2] . . . 1412
8.92.2.58 content_filter_hash_buckets() [1/2] . . . 1412
8.92.2.59 content_filter_hash_buckets() [2/2] . . . 1413
8.92.2.60 flow_controller_hash_buckets() [1/2] . . . 1413
8.92.2.61 flow_controller_hash_buckets() [2/2] . . . 1413
8.92.2.62 max_gather_destinations() [1/2] . . . 1413
8.92.2.63 max_gather_destinations() [2/2] . . . 1414
8.92.2.64 participant_user_data_max_length() [1/2] . . . 1414
8.92.2.65 participant_user_data_max_length() [2/2] . . . 1414

8.92.2.66 topic_data_max_length() [1/2] . . . 1414
8.92.2.67 topic_data_max_length() [2/2] . . . 1415
8.92.2.68 publisher_group_data_max_length() [1/2] . . . 1415
8.92.2.69 publisher_group_data_max_length() [2/2] . . . 1415
8.92.2.70 subscriber_group_data_max_length() [1/2] . . . 1415
8.92.2.71 subscriber_group_data_max_length() [2/2] . . . 1416
8.92.2.72 writer_user_data_max_length() [1/2] . . . 1416
8.92.2.73 writer_user_data_max_length() [2/2] . . . 1416
8.92.2.74 reader_user_data_max_length() [1/2] . . . 1416
8.92.2.75 reader_user_data_max_length() [2/2] . . . 1417
8.92.2.76 max_partitions() [1/2] . . . 1417
8.92.2.77 max_partitions() [2/2] . . . 1417
8.92.2.78 max_partition_cumulative_characters() . . . 1418
8.92.2.79 type_code_max_serialized_length() [1/2] . . . 1418
8.92.2.80 type_code_max_serialized_length() [2/2] . . . 1419
8.92.2.81 type_object_max_serialized_length() [1/2] . . . 1419
8.92.2.82 type_object_max_serialized_length() [2/2] . . . 1420
8.92.2.83 type_object_max_deserialized_length() [1/2] . . . 1420
8.92.2.84 type_object_max_deserialized_length() [2/2] . . . 1420
8.92.2.85 deserialized_type_object_dynamic_allocation_threshold() [1/2] . . . 1421
8.92.2.86 deserialized_type_object_dynamic_allocation_threshold() [2/2] . . . 1421
8.92.2.87 serialized_type_object_dynamic_allocation_threshold() [1/2] . . . 1422
8.92.2.88 serialized_type_object_dynamic_allocation_threshold() [2/2] . . . 1422
8.92.2.89 contentfilter_property_max_length() [1/2] . . . 1423
8.92.2.90 contentfilter_property_max_length() [2/2] . . . 1423
8.92.2.91 channel_seq_max_length() [1/2] . . . 1423
8.92.2.92 channel_seq_max_length() [2/2] . . . 1424
8.92.2.93 channel_filter_expression_max_length() [1/2] . . . 1424
8.92.2.94 channel_filter_expression_max_length() [2/2] . . . 1424
8.92.2.95 participant_property_list_max_length() [1/2] . . . 1424
8.92.2.96 participant_property_list_max_length() [2/2] . . . 1425
8.92.2.97 participant_property_string_max_length() [1/2] . . . 1425
8.92.2.98 participant_property_string_max_length() [2/2] . . . 1425
8.92.2.99 writer_property_list_max_length() [1/2] . . . 1425
8.92.2.100 writer_property_list_max_length() [2/2] . . . 1426
8.92.2.101 writer_property_string_max_length() [1/2] . . . 1426
8.92.2.102 writer_property_string_max_length() [2/2] . . . 1426
8.92.2.103 reader_property_list_max_length() [1/2] . . . 1426
8.92.2.104 reader_property_list_max_length() [2/2] . . . 1427

8.92.2.105 reader_property_string_max_length() [1/2] . . . 1427
8.92.2.106 reader_property_string_max_length() [2/2] . . . 1427
8.92.2.107 max_endpoint_groups() [1/2] . . . 1427
8.92.2.108 max_endpoint_groups() [2/2] . . . 1428
8.92.2.109 max_endpoint_group_cumulative_characters() [1/2] . . . 1428
8.92.2.110 max_endpoint_group_cumulative_characters() [2/2] . . . 1428
8.92.2.111 transport_info_list_max_length() [1/2] . . . 1428
8.92.2.112 transport_info_list_max_length() [2/2] . . . 1429
8.92.2.113 ignored_entity_replacement_kind() [1/2] . . . 1429
8.92.2.114 ignored_entity_replacement_kind() [2/2] . . . 1429
8.92.2.115 remote_topic_query_allocation() [1/2] . . . 1430
8.92.2.116 remote_topic_query_allocation() [2/2] . . . 1430
8.92.2.117 remote_topic_query_hash_buckets() [1/2] . . . 1430
8.92.2.118 remote_topic_query_hash_buckets() [2/2] . . . 1431
8.92.2.119 writer_data_tag_list_max_length() [1/2] . . . 1431
8.92.2.120 writer_data_tag_list_max_length() [2/2] . . . 1431
8.92.2.121 writer_data_tag_string_max_length() [1/2] . . . 1431
8.92.2.122 writer_data_tag_string_max_length() [2/2] . . . 1432
8.92.2.123 reader_data_tag_list_max_length() [1/2] . . . 1432
8.92.2.124 reader_data_tag_list_max_length() [2/2] . . . 1432
8.92.2.125 reader_data_tag_string_max_length() [1/2] . . . 1432
8.92.2.126 reader_data_tag_string_max_length() [2/2] . . . 1433
8.92.2.127 shmem_ref_transfer_mode_max_segments() [1/2] . . . 1433
8.92.2.128 shmem_ref_transfer_mode_max_segments() [2/2] . . . 1433
8.93 dds::core::policy::Durability Class Reference . . . 1433
8.93.1 Detailed Description . . . 1434
8.93.2 Usage . . . 1435
8.93.2.1 Transient and Persistent Durability . . . 1435
8.93.3 Compatibility . . . 1436
8.93.4 Constructor & Destructor Documentation . . . 1436
8.93.4.1 Durability() [1/2] . . . 1437
8.93.4.2 Durability() [2/2] . . . 1437
8.93.5 Member Function Documentation . . . 1437
8.93.5.1 kind() [1/2] . . . 1437
8.93.5.2 kind() [2/2] . . . 1438
8.93.5.3 Volatile() . . . 1438
8.93.5.4 TransientLocal() . . . 1438
8.93.5.5 Transient() . . . 1438
8.93.5.6 Persistent() . . . 1438

8.93.5.7 direct_communication() [1/2] . . . 1439
8.93.5.8 direct_communication() [2/2] . . . 1439
8.93.5.9 writer_depth() [1/2] . . . 1440
8.93.5.10 writer_depth() [2/2] . . . 1440
8.93.5.11 storage_settings() [1/3] . . . 1441
8.93.5.12 storage_settings() [2/3] . . . 1441
8.93.5.13 storage_settings() [3/3] . . . 1441
8.94 dds::core::policy::DurabilityKind_def Struct Reference . . . 1441
8.94.1 Detailed Description . . . 1442
8.94.2 Member Enumeration Documentation . . . 1442
8.94.2.1 type . . . 1442
8.95 dds::core::policy::DurabilityService Class Reference . . . 1443
8.95.1 Detailed Description . . . 1443
8.95.2 Usage . . . 1444
8.95.3 Constructor & Destructor Documentation . . . 1444
8.95.3.1 DurabilityService() [1/2] . . . 1444
8.95.3.2 DurabilityService() [2/2] . . . 1444
8.95.4 Member Function Documentation . . . 1445
8.95.4.1 service_cleanup_delay() [1/2] . . . 1445
8.95.4.2 service_cleanup_delay() [2/2] . . . 1445
8.95.4.3 history_kind() [1/2] . . . 1445
8.95.4.4 history_kind() [2/2] . . . 1446
8.95.4.5 history_depth() [1/2] . . . 1446
8.95.4.6 history_depth() [2/2] . . . 1446
8.95.4.7 max_samples() [1/2] . . . 1446
8.95.4.8 max_samples() [2/2] . . . 1447
8.95.4.9 max_instances() . . . 1447
8.95.4.10 max_samples_per_instance() [1/2] . . . 1447
8.95.4.11 max_samples_per_instance() [2/2] . . . 1447
8.96 dds::core::Duration Class Reference . . . 1448
8.96.1 Detailed Description . . . 1449
8.96.2 Constructor & Destructor Documentation . . . 1450
8.96.2.1 Duration() [1/3] . . . 1450
8.96.2.2 Duration() [2/3] . . . 1450
8.96.2.3 Duration() [3/3] . . . 1450
8.96.3 Member Function Documentation . . . 1450
8.96.3.1 zero() . . . 1451
8.96.3.2 infinite() . . . 1451
8.96.3.3 automatic() . . . 1451

8.96.3.4 from_microsecs() . . . 1451
8.96.3.5 from_millisecs() . . . 1452
8.96.3.6 from_secs() . . . 1452
8.96.3.7 sec() [1/2] . . . 1452
8.96.3.8 sec() [2/2] . . . 1453
8.96.3.9 nanosec() [1/2] . . . 1453
8.96.3.10 nanosec() [2/2] . . . 1453
8.96.3.11 compare() . . . 1453
8.96.3.12 operator>() . . . 1454
8.96.3.13 operator>=() . . . 1454
8.96.3.14 operator==() . . . 1455
8.96.3.15 operator"!=() . . . 1455
8.96.3.16 operator<=() . . . 1455
8.96.3.17 operator<() . . . 1456
8.96.3.18 operator+=() . . . 1456
8.96.3.19 operator-=() . . . 1456
8.96.3.20 operator+() . . . 1457
8.96.3.21 operator-() . . . 1457
8.96.3.22 to_nanosecs() . . . 1458
8.96.3.23 to_millisecs() . . . 1458
8.96.3.24 to_microsecs() . . . 1458
8.96.3.25 to_secs() . . . 1458
8.96.3.26 to_chrono() . . . 1459
8.96.4 Friends And Related Function Documentation . . . 1459
8.96.4.1 operator*() [1/2] . . . 1459
8.96.4.2 operator*() [2/2] . . . 1459
8.96.4.3 operator/() . . . 1460
8.96.4.4 swap() . . . 1460
8.97 rti::topic::dynamic_type< TopicType > Struct Template Reference . . . 1461
8.97.1 Detailed Description . . . 1461
8.98 dds::core::xtypes::DynamicData Class Reference . . . 1461
8.98.1 Detailed Description . . . 1464
8.98.2 Member Names and Indexes . . . 1465
8.98.2.1 Hierarchical Member Names . . . 1465
8.98.3 Exceptions accessing a member . . . 1466
8.98.3.1 Performance Considerations . . . 1466
8.98.4 Constructor & Destructor Documentation . . . 1467
8.98.4.1 DynamicData() [1/2] . . . 1467
8.98.4.2 DynamicData() [2/2] . . . 1468

8.98.5 Member Function Documentation 1468
8.98.5.1 value() [1/4] 1468
8.98.5.2 value() [2/4] 1469
8.98.5.3 value() [3/4] 1470
8.98.5.4 value() [4/4] 1471
8.98.5.5 get_values() [1/4] 1471
8.98.5.6 get_values() [2/4] 1472
8.98.5.7 get_values() [3/4] 1473
8.98.5.8 get_values() [4/4] 1473
8.98.5.9 set_values() [1/2] 1474
8.98.5.10 set_values() [2/2] 1474
8.98.5.11 loan_value() [1/4] 1475
8.98.5.12 loan_value() [2/4] 1476
8.98.5.13 loan_value() [3/4] 1476
8.98.5.14 loan_value() [4/4] 1477
8.98.5.15 discriminator_value() 1477
8.98.5.16 get_discriminator() 1478
8.98.5.17 set_discriminator() 1479
8.98.5.18 clear_all_members() 1479
8.98.5.19 clear_optional_member() [1/2] 1480
8.98.5.20 clear_optional_member() [2/2] 1480
8.98.5.21 clear_member() [1/2] 1480
8.98.5.22 clear_member() [2/2] 1481
8.98.5.23 is_cdr() 1481
8.98.5.24 get_cdr_buffer() 1482
8.98.5.25 set_cdr_buffer() 1482
8.98.5.26 type() 1483
8.98.5.27 type_kind() 1483
8.98.5.28 member_count() 1483
8.98.5.29 member_exists() [1/2] 1483
8.98.5.30 member_exists() [2/2] 1484
8.98.5.31 member_exists_in_type() [1/2] 1484
8.98.5.32 member_exists_in_type() [2/2] 1484
8.98.5.33 info() 1484
8.98.5.34 member_info() [1/2] 1485
8.98.5.35 member_info() [2/2] 1486
8.98.5.36 member_index() 1487
8.98.5.37 is_member_key() [1/2] 1488
8.98.5.38 is_member_key() [2/2] 1488

8.98.5.39 member_type() [1/2] . . . 1489
8.98.5.40 member_type() [2/2] . . . 1489
8.98.5.41 from_string() . . . 1489
8.98.6 Friends And Related Function Documentation . . . 1490
8.98.6.1 to_cdr_buffer() . . . 1490
8.98.6.2 from_cdr_buffer() . . . 1490
8.98.6.3 convert() [1/2] . . . 1491
8.98.6.4 convert() [2/2] . . . 1492
8.98.6.5 can_convert() . . . 1492
8.98.6.6 get_tuple() . . . 1493
8.98.6.7 set_tuple() . . . 1494
8.99 rti::core::xtypes::DynamicDataInfo Class Reference . . . 1494
8.99.1 Detailed Description . . . 1494
8.99.2 Member Function Documentation . . . 1494
8.99.2.1 member_count() . . . 1495
8.99.2.2 stored_size() . . . 1495
8.100 rti::core::xtypes::DynamicDataMemberInfo Class Reference . . . 1495
8.100.1 Detailed Description . . . 1495
8.100.2 Member Function Documentation . . . 1496
8.100.2.1 member_index() . . . 1496
8.100.2.2 member_name() . . . 1496
8.100.2.3 member_kind() . . . 1496
8.100.2.4 element_count() . . . 1497
8.100.2.5 element_kind() . . . 1497
8.100.2.6 member_exists() . . . 1497
8.101 rti::core::xtypes::DynamicDataProperty Class Reference . . . 1497
8.101.1 Detailed Description . . . 1498
8.101.2 Member Function Documentation . . . 1498
8.101.2.1 buffer_initial_size() [1/2] . . . 1498
8.101.2.2 buffer_initial_size() [2/2] . . . 1498
8.101.2.3 buffer_max_size() [1/2] . . . 1499
8.101.2.4 buffer_max_size() [2/2] . . . 1499
8.102 rti::core::xtypes::DynamicDataTypeSerializationProperty Class Reference . . . 1500
8.102.1 Detailed Description . . . 1500
8.102.2 Constructor & Destructor Documentation . . . 1501
8.102.2.1 DynamicDataTypeSerializationProperty() [1/2] . . . 1501
8.102.2.2 DynamicDataTypeSerializationProperty() [2/2] . . . 1501
8.102.3 Member Function Documentation . . . 1501
8.102.3.1 max_size_serialized() [1/2] . . . 1501

8.102.3.2 max_size_serialized() [2/2] . . . 1501
8.102.3.3 min_size_serialized() [1/2] . . . 1502
8.102.3.4 min_size_serialized() [2/2] . . . 1502
8.102.3.5 trim_to_size() [1/2] . . . 1502
8.102.3.6 trim_to_size() [2/2] . . . 1502
8.102.3.7 skip_deserialization() [1/2] . . . 1503
8.102.3.8 skip_deserialization() [2/2] . . . 1503
8.102.4 Member Data Documentation . . . 1503
8.102.4.1 DEFAULT . . . 1503
8.103 dds::core::xtypes::DynamicType Class Reference . . . 1504
8.103.1 Detailed Description . . . 1505
8.103.2 Member Function Documentation . . . 1506
8.103.2.1 kind() . . . 1506
8.103.2.2 name() . . . 1506
8.103.2.3 is_assignable_from() . . . 1506
8.103.3 Friends And Related Function Documentation . . . 1507
8.103.3.1 is_primitive_type() . . . 1507
8.103.3.2 is_constructed_type() . . . 1507
8.103.3.3 is_collection_type() . . . 1507
8.103.3.4 is_aggregation_type() . . . 1507
8.103.3.5 operator<<() . . . 1508
8.103.3.6 print_idl() . . . 1508
8.103.3.7 to_string() [1/2] . . . 1508
8.103.3.8 to_string() [2/2] . . . 1508
8.104 rti::core::xtypes::DynamicTypePrintFormatProperty Class Reference . . . 1509
8.104.1 Detailed Description . . . 1510
8.104.2 Constructor & Destructor Documentation . . . 1510
8.104.2.1 DynamicTypePrintFormatProperty() . . . 1510
8.104.3 Member Function Documentation . . . 1510
8.104.3.1 indent() [1/2] . . . 1510
8.104.3.2 indent() [2/2] . . . 1511
8.104.3.3 print_ordinals() [1/2] . . . 1511
8.104.3.4 print_ordinals() [2/2] . . . 1512
8.104.3.5 print_kind() [1/2] . . . 1512
8.104.3.6 print_kind() [2/2] . . . 1513
8.104.3.7 print_complete_type() [1/2] . . . 1513
8.104.3.8 print_complete_type() [2/2] . . . 1514
8.105 rti::core::EndpointGroup Class Reference . . . 1515
8.105.1 Detailed Description . . . 1515

8.105.2 Constructor & Destructor Documentation . . . 1515
8.105.2.1 EndpointGroup() . . . 1515
8.105.3 Member Function Documentation . . . 1515
8.105.3.1 role_name() [1/2] . . . 1516
8.105.3.2 role_name() [2/2] . . . 1516
8.105.3.3 quorum_count() [1/2] . . . 1516
8.105.3.4 quorum_count() [2/2] . . . 1516
8.106 rti::topic::trust::EndpointTrustAlgorithmInfo Class Reference . . . 1516
8.106.1 Detailed Description . . . 1517
8.106.2 Constructor & Destructor Documentation . . . 1517
8.106.2.1 EndpointTrustAlgorithmInfo() . . . 1517
8.106.3 Member Function Documentation . . . 1517
8.106.3.1 interceptor() . . . 1517
8.107 rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo Class Reference . . . 1517
8.107.1 Detailed Description . . . 1518
8.107.2 Constructor & Destructor Documentation . . . 1518
8.107.2.1 EndpointTrustInterceptorAlgorithmInfo() . . . 1518
8.107.3 Member Function Documentation . . . 1518
8.107.3.1 required_mask() . . . 1518
8.107.3.2 supported_mask() . . . 1518
8.108 rti::topic::trust::EndpointTrustProtectionInfo Class Reference . . . 1519
8.108.1 Detailed Description . . . 1519
8.108.2 Constructor & Destructor Documentation . . . 1519
8.108.2.1 EndpointTrustProtectionInfo() . . . 1519
8.108.3 Member Function Documentation . . . 1519
8.108.3.1 bitmask() . . . 1520
8.108.3.2 plugin_bitmask() . . . 1520
8.109 dds::core::Entity Class Reference . . . 1520
8.109.1 Detailed Description . . . 1521
8.109.2 Abstract operations . . . 1522
8.109.2.1 set_qos (abstract) . . . 1522
8.109.2.2 get_qos (abstract) . . . 1523
8.109.2.3 set_listener (abstract) . . . 1523
8.109.2.4 get_listener (abstract) . . . 1524
8.109.3 Member Function Documentation . . . 1524
8.109.3.1 enable() . . . 1524
8.109.3.2 status_changes() . . . 1525
8.109.3.3 instance_handle() . . . 1525
8.109.3.4 close() . . . 1526

8.109.3.5 retain() . . . 1526
8.109.4 Friends And Related Function Documentation . . . 1526
8.109.4.1 polymorphic_cast() . . . 1526
8.110 dds::core::policy::EntityFactory Class Reference . . . 1527
8.110.1 Detailed Description . . . 1528
8.110.2 Usage . . . 1528
8.110.3 Member Function Documentation . . . 1528
8.110.3.1 TEntityFactory() [1/2] . . . 1529
8.110.3.2 TEntityFactory() [2/2] . . . 1529
8.110.3.3 autoenable_created_entities() [1/2] . . . 1529
8.110.3.4 autoenable_created_entities() [2/2] . . . 1529
8.110.3.5 AutoEnable() . . . 1529
8.110.3.6 ManuallyEnable() . . . 1529
8.111 rti::core::policy::EntityName Class Reference . . . 1530
8.111.1 Detailed Description . . . 1530
8.111.2 Usage . . . 1530
8.111.3 Constructor & Destructor Documentation . . . 1531
8.111.3.1 EntityName() [1/2] . . . 1531
8.111.3.2 EntityName() [2/2] . . . 1531
8.111.4 Member Function Documentation . . . 1531
8.111.4.1 name() [1/3] . . . 1531
8.111.4.2 name() [2/3] . . . 1531
8.111.4.3 name() [3/3] . . . 1532
8.111.4.4 role_name() [1/3] . . . 1532
8.111.4.5 role_name() [2/3] . . . 1532
8.111.4.6 role_name() [3/3] . . . 1533
8.112 dds::core::xtypes::EnumMember Class Reference . . . 1533
8.112.1 Detailed Description . . . 1533
8.112.2 Constructor & Destructor Documentation . . . 1533
8.112.2.1 EnumMember() . . . 1534
8.112.3 Member Function Documentation . . . 1534
8.112.3.1 name() [1/3] . . . 1534
8.112.3.2 name() [2/3] . . . 1534
8.112.3.3 name() [3/3] . . . 1534
8.112.3.4 ordinal() [1/2] . . . 1534
8.112.3.5 ordinal() [2/2] . . . 1535
8.113 dds::core::xtypes::EnumType Class Reference . . . 1535
8.113.1 Detailed Description . . . 1536
8.113.2 Constructor & Destructor Documentation . . . 1536

8.113.2.1 EnumType() [1/4] . . . 1536
8.113.2.2 EnumType() [2/4] . . . 1536
8.113.2.3 EnumType() [3/4] . . . 1537
8.113.2.4 EnumType() [4/4] . . . 1537
8.113.3 Member Function Documentation . . . 1538
8.113.3.1 find_member_by_ordinal() . . . 1538
8.113.3.2 add_member() . . . 1538
8.113.3.3 add_members() [1/3] . . . 1538
8.113.3.4 add_members() [2/3] . . . 1538
8.113.3.5 add_members() [3/3] . . . 1539
8.114 rti::test::EnvVarToken Class Reference . . . 1539
8.114.1 Detailed Description . . . 1539
8.115 dds::core::Error Class Reference . . . 1539
8.115.1 Detailed Description . . . 1540
8.115.2 Constructor & Destructor Documentation . . . 1540
8.115.2.1 Error() [1/2] . . . 1540
8.115.2.2 Error() [2/2] . . . 1540
8.115.3 Member Function Documentation . . . 1540
8.115.3.1 what() . . . 1540
8.116 rti::core::policy::Event Class Reference . . . 1541
8.116.1 Detailed Description . . . 1541
8.116.2 Constructor & Destructor Documentation . . . 1542
8.116.2.1 Event() [1/2] . . . 1542
8.116.2.2 Event() [2/2] . . . 1542
8.116.3 Member Function Documentation . . . 1542
8.116.3.1 thread() [1/3] . . . 1542
8.116.3.2 thread() [2/3] . . . 1543
8.116.3.3 thread() [3/3] . . . 1543
8.116.3.4 initial_count() [1/2] . . . 1543
8.116.3.5 initial_count() [2/2] . . . 1543
8.116.3.6 max_count() [1/2] . . . 1544
8.116.3.7 max_count() [2/2] . . . 1544
8.117 rti::core::status::EventCount< IntegerType > Class Template Reference . . . 1544
8.117.1 Detailed Description . . . 1545
8.117.2 Member Function Documentation . . . 1545
8.117.2.1 total() . . . 1545
8.117.2.2 change() . . . 1545
8.117.3 Friends And Related Function Documentation . . . 1545
8.117.3.1 operator<<() . . . 1546

8.118 dds::core::Exception Class Reference . . . 1546
8.118.1 Detailed Description . . . 1547
8.118.2 Member Function Documentation . . . 1547
8.118.2.1 what() . . . 1547
8.119 rti::topic::ExpressionProperty Class Reference . . . 1547
8.119.1 Detailed Description . . . 1548
8.119.2 Constructor & Destructor Documentation . . . 1548
8.119.2.1 ExpressionProperty() [1/2] . . . 1548
8.119.2.2 ExpressionProperty() [2/2] . . . 1548
8.119.3 Member Function Documentation . . . 1548
8.119.3.1 key_only_filter() [1/2] . . . 1548
8.119.3.2 key_only_filter() [2/2] . . . 1549
8.119.3.3 writer_side_filter_optimization() [1/2] . . . 1549
8.119.3.4 writer_side_filter_optimization() [2/2] . . . 1549
8.120 rti::topic::extensibility< TopicType > Struct Template Reference . . . 1550
8.120.1 Detailed Description . . . 1550
8.121 dds::core::xtypes::ExtensibilityKind_def Struct Reference . . . 1550
8.121.1 Detailed Description . . . 1550
8.121.2 Member Enumeration Documentation . . . 1551
8.121.2.1 type . . . 1551
8.122 dds::core::external< T > Class Template Reference . . . 1551
8.122.1 Detailed Description . . . 1552
8.122.2 Constructor & Destructor Documentation . . . 1553
8.122.2.1 external() [1/4] . . . 1553
8.122.2.2 external() [2/4] . . . 1553
8.122.2.3 external() [3/4] . . . 1554
8.122.2.4 external() [4/4] . . . 1554
8.122.2.5 ∼external() . . . 1554
8.122.3 Member Function Documentation . . . 1555
8.122.3.1 operator=() . . . 1555
8.122.3.2 operator∗() [1/2] . . . 1555
8.122.3.3 operator∗() [2/2] . . . 1555
8.122.3.4 get() [1/2] . . . 1556
8.122.3.5 get() [2/2] . . . 1556
8.122.3.6 get_shared_ptr() . . . 1556
8.122.3.7 operator->() [1/2] . . . 1556
8.122.3.8 operator->() [2/2] . . . 1557
8.122.3.9 operator==() . . . 1557
8.122.3.10 operator"!=() . . . 1557

8.122.3.11 operator bool() . . . 1557
8.122.3.12 is_locked() . . . 1558
8.122.4 Friends And Related Function Documentation . . . 1558
8.122.4.1 swap . . . 1558
8.123 dds::topic::Filter Class Reference . . . 1558
8.123.1 Detailed Description . . . 1559
8.123.2 Constructor & Destructor Documentation . . . 1559
8.123.2.1 Filter() [1/3] . . . 1559
8.123.2.2 Filter() [2/3] . . . 1559
8.123.2.3 Filter() [3/3] . . . 1560
8.123.3 Member Function Documentation . . . 1560
8.123.3.1 expression() . . . 1560
8.123.3.2 begin() [1/2] . . . 1561
8.123.3.3 end() [1/2] . . . 1561
8.123.3.4 begin() [2/2] . . . 1561
8.123.3.5 end() [2/2] . . . 1561
8.123.3.6 parameters() . . . 1561
8.123.3.7 add_parameter() . . . 1562
8.123.3.8 parameters_length() . . . 1562
8.123.3.9 name() [1/2] . . . 1562
8.123.3.10 name() [2/2] . . . 1562
8.124 rti::topic::FilterSampleInfo Class Reference . . . 1562
8.124.1 Detailed Description . . . 1563
8.124.2 Member Function Documentation . . . 1563
8.124.2.1 related_sample_identity() . . . 1563
8.124.2.2 related_source_guid() . . . 1564
8.124.2.3 related_reader_guid() . . . 1564
8.124.2.4 sample_identity() . . . 1564
8.124.2.5 source_guid() . . . 1565
8.124.2.6 priority() . . . 1565
8.125 rti::flat::FinalAlignedArrayOffset< ElementOffset, N > Class Template Reference . . . 1565
8.125.1 Detailed Description . . . 1566
8.125.2 Member Function Documentation . . . 1566
8.125.2.1 get_element() . . . 1566
8.126 rti::flat::FinalArrayOffset< ElementOffset, N > Class Template Reference . . . 1567
8.126.1 Detailed Description . . . 1567
8.126.2 Member Function Documentation . . . 1568
8.126.2.1 get_element() . . . 1568
8.127 rti::flat::FinalOffset< T > Class Template Reference . . . 1568

8.127.1 Detailed Description . . . 1569
8.128 rti::flat::FinalSequenceBuilder< ElementOffset > Class Template Reference . . . 1569
8.128.1 Detailed Description . . . 1569
8.128.2 Member Function Documentation . . . 1570
8.128.2.1 add_next() . . . 1570
8.128.2.2 add_n() . . . 1570
8.128.2.3 finish() . . . 1571
8.129 rti::flat::flat_type_traits< T > Struct Template Reference . . . 1571
8.129.1 Detailed Description . . . 1571
8.130 rti::pub::FlowController Class Reference . . . 1572
8.130.1 Detailed Description . . . 1573
8.130.2 Constructor & Destructor Documentation . . . 1573
8.130.2.1 FlowController() . . . 1573
8.130.3 Member Function Documentation . . . 1574
8.130.3.1 name() . . . 1574
8.130.3.2 participant() . . . 1574
8.130.3.3 property() [1/2] . . . 1574
8.130.3.4 property() [2/2] . . . 1575
8.130.3.5 trigger_flow() . . . 1575
8.130.3.6 retain() . . . 1576
8.130.3.7 close() . . . 1576
8.130.3.8 closed() . . . 1576
8.130.4 Friends And Related Function Documentation . . . 1576
8.130.4.1 find_flow_controller() . . . 1576
8.131 rti::pub::FlowControllerProperty Class Reference . . . 1577
8.131.1 Detailed Description . . . 1578
8.131.2 Constructor & Destructor Documentation . . . 1578
8.131.2.1 FlowControllerProperty() [1/2] . . . 1578
8.131.2.2 FlowControllerProperty() [2/2] . . . 1579
8.131.3 Member Function Documentation . . . 1579
8.131.3.1 scheduling_policy() [1/2] . . . 1579
8.131.3.2 scheduling_policy() [2/2] . . . 1579
8.131.3.3 token_bucket() [1/2] . . . 1579
8.131.3.4 token_bucket() [2/2] . . . 1580
8.131.3.5 RoundRobin() . . . 1580
8.131.3.6 EarliestDeadlineFirst() . . . 1580
8.131.3.7 HighestPriorityFirst() . . . 1580
8.132 rti::pub::FlowControllerSchedulingPolicy_def Struct Reference . . . 1581
8.132.1 Detailed Description . . . 1581

8.132.2 Member Enumeration Documentation . . . 1581
8.132.2.1 type . . . 1581
8.133 rti::pub::FlowControllerTokenBucketProperty Class Reference . . . 1583
8.133.1 Detailed Description . . . 1584
8.133.2 Constructor & Destructor Documentation . . . 1584
8.133.2.1 FlowControllerTokenBucketProperty() . . . 1585
8.133.3 Member Function Documentation . . . 1585
8.133.3.1 max_tokens() [1/2] . . . 1585
8.133.3.2 max_tokens() [2/2] . . . 1585
8.133.3.3 tokens_added_per_period() [1/2] . . . 1586
8.133.3.4 tokens_added_per_period() [2/2] . . . 1586
8.133.3.5 tokens_leaked_per_period() [1/2] . . . 1586
8.133.3.6 tokens_leaked_per_period() [2/2] . . . 1587
8.133.3.7 period() [1/2] . . . 1587
8.133.3.8 period() [2/2] . . . 1587
8.133.3.9 bytes_per_token() [1/2] . . . 1588
8.133.3.10 bytes_per_token() [2/2] . . . 1588
8.134 Foo Class Reference . . . 1588
8.134.1 Detailed Description . . . 1589
8.135 dds::sub::GenerationCount Class Reference . . . 1589
8.135.1 Detailed Description . . . 1589
8.135.2 Constructor & Destructor Documentation . . . 1589
8.135.2.1 GenerationCount() [1/2] . . . 1589
8.135.2.2 GenerationCount() [2/2] . . . 1590
8.135.3 Member Function Documentation . . . 1590
8.135.3.1 disposed() . . . 1590
8.135.3.2 no_writers() . . . 1590
8.136 dds::core::policy::GroupData Class Reference . . . 1591
8.136.1 Detailed Description . . . 1591
8.136.2 Usage . . . 1592
8.136.3 Constructor & Destructor Documentation . . . 1592
8.136.3.1 GroupData() [1/3] . . . 1592
8.136.3.2 GroupData() [2/3] . . . 1592
8.136.3.3 GroupData() [3/3] . . . 1593
8.136.4 Member Function Documentation . . . 1593
8.136.4.1 value() [1/3] . . . 1593
8.136.4.2 value() [2/3] . . . 1593
8.136.4.3 value() [3/3] . . . 1593
8.136.4.4 begin() . . . 1593

8.136.4.5 end() . . . 1594
8.137 dds::core::cond::GuardCondition Class Reference . . . 1594
8.137.1 Detailed Description . . . 1594
8.137.2 Member Function Documentation . . . 1594
8.137.2.1 handler() . . . 1595
8.137.2.2 reset_handler() . . . 1595
8.137.2.3 trigger_value() . . . 1595
8.138 rti::core::Guid Class Reference . . . 1596
8.138.1 Detailed Description . . . 1597
8.138.2 Constructor & Destructor Documentation . . . 1597
8.138.2.1 Guid() . . . 1597
8.138.3 Member Function Documentation . . . 1597
8.138.3.1 unknown() . . . 1597
8.138.3.2 zero() . . . 1597
8.138.3.3 automatic() . . . 1597
8.138.3.4 operator[]() [1/2] . . . 1598
8.138.3.5 operator[]() [2/2] . . . 1598
8.138.3.6 operator<() . . . 1598
8.138.3.7 operator<=() . . . 1598
8.138.3.8 operator>() . . . 1598
8.138.3.9 operator>=() . . . 1599
8.138.4 Friends And Related Function Documentation . . . 1599
8.138.4.1 operator<<() . . . 1599
8.138.5 Member Data Documentation . . . 1599
8.138.5.1 LENGTH . . . 1599
8.139 rti::util::heap_monitoring::HeapMonitoringParams Class Reference . . . 1599
8.139.1 Detailed Description . . . 1600
8.139.2 Constructor & Destructor Documentation . . . 1600
8.139.2.1 HeapMonitoringParams() . . . 1600
8.139.3 Member Function Documentation . . . 1600
8.139.3.1 snapshot_output_format() [1/2] . . . 1601
8.139.3.2 snapshot_output_format() [2/2] . . . 1601
8.139.3.3 snapshot_content_format() [1/2] . . . 1601
8.139.3.4 snapshot_content_format() [2/2] . . . 1601
8.140 dds::core::policy::History Class Reference . . . 1602
8.140.1 Detailed Description . . . 1602
8.140.2 Usage . . . 1603
8.140.3 Consistency . . . 1604
8.140.4 Constructor & Destructor Documentation . . . 1604

8.140.4.1 History() [1/2] . . . 1604
8.140.4.2 History() [2/2] . . . 1604
8.140.5 Member Function Documentation . . . 1604
8.140.5.1 kind() [1/2] . . . 1604
8.140.5.2 kind() [2/2] . . . 1605
8.140.5.3 depth() [1/2] . . . 1605
8.140.5.4 depth() [2/2] . . . 1605
8.140.5.5 KeepAll() . . . 1606
8.140.5.6 KeepLast() . . . 1606
8.141 dds::core::policy::HistoryKind_def Struct Reference . . . 1606
8.141.1 Detailed Description . . . 1606
8.141.2 Member Enumeration Documentation . . . 1606
8.141.2.1 type . . . 1606
8.142 rti::core::policy::IgnoredEntityReplacementKind_def Struct Reference . . . 1607
8.142.1 Detailed Description . . . 1607
8.142.2 Member Enumeration Documentation . . . 1607
8.142.2.1 type . . . 1608
8.143 dds::core::IllegalOperationError Class Reference . . . 1608
8.143.1 Detailed Description . . . 1608
8.143.2 Member Function Documentation . . . 1608
8.143.2.1 what() . . . 1609
8.144 dds::core::ImmutablePolicyError Class Reference . . . 1609
8.144.1 Detailed Description . . . 1609
8.144.2 Member Function Documentation . . . 1609
8.144.2.1 what() . . . 1610
8.145 dds::core::InconsistentPolicyError Class Reference . . . 1610
8.145.1 Detailed Description . . . 1610
8.145.2 Member Function Documentation . . . 1610
8.145.2.1 what() . . . 1611
8.146 dds::core::status::InconsistentTopicStatus Class Reference . . . 1611
8.146.1 Detailed Description . . . 1611
8.146.2 Member Function Documentation . . . 1612
8.146.2.1 total_count() . . . 1612
8.146.2.2 total_count_change() . . . 1612
8.147 dds::core::InstanceHandle Class Reference . . . 1612
8.147.1 Detailed Description . . . 1613
8.147.2 Constructor & Destructor Documentation . . . 1613
8.147.2.1 InstanceHandle() . . . 1613
8.147.3 Member Function Documentation . . . 1613

8.147.3.1 operator==() . . . 1614
8.147.3.2 nil() . . . 1614
8.147.3.3 is_nil() . . . 1614
8.147.4 Friends And Related Function Documentation . . . 1614
8.147.4.1 operator<<() . . . 1614
8.148 dds::sub::status::InstanceState Class Reference . . . 1615
8.148.1 Detailed Description . . . 1616
8.148.2 Member Typedef Documentation . . . 1616
8.148.2.1 MaskType . . . 1617
8.148.3 Constructor & Destructor Documentation . . . 1617
8.148.3.1 InstanceState() . . . 1617
8.148.4 Member Function Documentation . . . 1617
8.148.4.1 alive() . . . 1617
8.148.4.2 not_alive_disposed() . . . 1617
8.148.4.3 not_alive_no_writers() . . . 1618
8.148.4.4 not_alive_mask() . . . 1618
8.148.4.5 any() . . . 1618
8.148.5 Friends And Related Function Documentation . . . 1618
8.148.5.1 operator<<() . . . 1618
8.149 dds::core::InvalidArgumentError Class Reference . . . 1619
8.149.1 Detailed Description . . . 1619
8.149.2 Member Function Documentation . . . 1619
8.149.2.1 what() . . . 1619
8.150 dds::core::InvalidDowncastError Class Reference . . . 1620
8.150.1 Detailed Description . . . 1620
8.150.2 Member Function Documentation . . . 1620
8.150.2.1 what() . . . 1620
8.151 rti::core::status::InvalidLocalAdvanceNoticeStatus Class Reference . . . 1621
8.151.1 Detailed Description . . . 1621
8.151.2 Member Function Documentation . . . 1621
8.151.2.1 expiration_time() . . . 1621
8.151.2.2 expiration_code() . . . 1621
8.151.2.3 expiration_message() . . . 1622
8.152 dds::topic::is_topic_type< T > Struct Template Reference . . . 1622
8.152.1 Detailed Description . . . 1622
8.153 rti::request::IsReplyRelatedPredicate< T > Class Template Reference . . . 1622
8.153.1 Detailed Description . . . 1623
8.153.2 Constructor & Destructor Documentation . . . 1623
8.153.2.1 IsReplyRelatedPredicate() . . . 1623

8.153.3 Member Function Documentation . . . 1623
8.153.3.1 operator()() [1/3] . . . 1623
8.153.3.2 operator()() [2/3] . . . 1624
8.153.3.3 operator()() [3/3] . . . 1624
8.154 rti::sub::IsValidData< T > Struct Template Reference . . . 1624
8.154.1 Detailed Description . . . 1625
8.154.2 Member Typedef Documentation . . . 1625
8.154.2.1 sample_type . . . 1625
8.154.3 Member Function Documentation . . . 1625
8.154.3.1 operator()() . . . 1625
8.155 dds::core::KeyedBytesTopicType Class Reference . . . 1626
8.155.1 Detailed Description . . . 1626
8.155.2 Constructor & Destructor Documentation . . . 1626
8.155.2.1 KeyedBytesTopicType() [1/2] . . . 1626
8.155.2.2 KeyedBytesTopicType() [2/2] . . . 1627
8.155.3 Member Function Documentation . . . 1627
8.155.3.1 key() [1/3] . . . 1627
8.155.3.2 key() [2/3] . . . 1627
8.155.3.3 key() [3/3] . . . 1627
8.155.3.4 operator std::vector< uint8_t >() . . . 1627
8.155.3.5 value() [1/2] . . . 1628
8.155.3.6 value() [2/2] . . . 1628
8.155.3.7 operator[]() [1/2] . . . 1628
8.155.3.8 operator[]() [2/2] . . . 1628
8.155.3.9 length() . . . 1628
8.156 dds::core::KeyedStringTopicType Class Reference . . . 1629
8.156.1 Detailed Description . . . 1629
8.156.2 Constructor & Destructor Documentation . . . 1629
8.156.2.1 KeyedStringTopicType() [1/2] . . . 1629
8.156.2.2 KeyedStringTopicType() [2/2] . . . 1630
8.156.3 Member Function Documentation . . . 1630
8.156.3.1 key() [1/3] . . . 1630
8.156.3.2 key() [2/3] . . . 1630
8.156.3.3 key() [3/3] . . . 1630
8.156.3.4 value() [1/3] . . . 1631
8.156.3.5 value() [2/3] . . . 1631
8.156.3.6 value() [3/3] . . . 1631
8.157 dds::core::policy::LatencyBudget Class Reference . . . 1631
8.157.1 Detailed Description . . . 1632

8.157.2 Usage . . . 1632
8.157.3 Compatibility . . . 1632
8.157.4 Constructor & Destructor Documentation . . . 1633
8.157.4.1 LatencyBudget() [1/2] . . . 1633
8.157.4.2 LatencyBudget() [2/2] . . . 1633
8.157.5 Member Function Documentation . . . 1633
8.157.5.1 duration() [1/2] . . . 1633
8.157.5.2 duration() [2/2] . . . 1633
8.158 rti::config::LibraryVersion Class Reference . . . 1634
8.158.1 Detailed Description . . . 1634
8.158.2 Member Function Documentation . . . 1634
8.158.2.1 major_version() . . . 1634
8.158.2.2 minor_version() . . . 1634
8.158.2.3 release_version() . . . 1635
8.158.2.4 build_version() . . . 1635
8.159 dds::core::policy::Lifespan Class Reference . . . 1635
8.159.1 Detailed Description . . . 1635
8.159.2 Usage . . . 1636
8.159.3 Constructor & Destructor Documentation . . . 1636
8.159.3.1 Lifespan() [1/2] . . . 1636
8.159.3.2 Lifespan() [2/2] . . . 1637
8.159.4 Member Function Documentation . . . 1637
8.159.4.1 duration() [1/2] . . . 1637
8.159.4.2 duration() [2/2] . . . 1637
8.160 Listener Class Reference . . . 1637
8.160.1 Detailed Description . . . 1638
8.160.2 Access to Plain Communication Status . . . 1638
8.160.3 Access to Read Communication Status . . . 1639
8.160.4 How to set a listener . . . 1639
8.160.5 Operations Allowed in Listener Callbacks . . . 1639
8.160.6 Best Practices with Listeners . . . 1640
8.161 rti::core::ListenerBinder< Entity, Listener > Class Template Reference . . . 1641
8.161.1 Detailed Description . . . 1642
8.161.2 Member Function Documentation . . . 1643
8.161.2.1 get() [1/2] . . . 1643
8.161.2.2 get() [2/2] . . . 1643
8.161.2.3 listener() [1/2] . . . 1643
8.161.2.4 listener() [2/2] . . . 1643
8.161.2.5 entity() . . . 1644

8.161.3 Friends And Related Function Documentation . . . 1644
8.161.3.1 bind_listener() [1/2] . . . 1644
8.161.3.2 bind_listener() [2/2] . . . 1644
8.161.3.3 bind_and_manage_listener() [1/2] . . . 1645
8.161.3.4 bind_and_manage_listener() [2/2] . . . 1645
8.162 dds::core::policy::Liveliness Class Reference . . . 1645
8.162.1 Detailed Description . . . 1646
8.162.2 Usage . . . 1647
8.162.3 Compatibility . . . 1648
8.162.4 Constructor & Destructor Documentation . . . 1648
8.162.4.1 Liveliness() [1/2] . . . 1648
8.162.4.2 Liveliness() [2/2] . . . 1649
8.162.5 Member Function Documentation . . . 1649
8.162.5.1 kind() [1/2] . . . 1649
8.162.5.2 kind() [2/2] . . . 1649
8.162.5.3 lease_duration() [1/2] . . . 1650
8.162.5.4 lease_duration() [2/2] . . . 1650
8.162.5.5 Automatic() . . . 1650
8.162.5.6 ManualByParticipant() . . . 1651
8.162.5.7 ManualByTopic() . . . 1651
8.162.5.8 assertions_per_lease_duration() [1/2] . . . 1651
8.162.5.9 assertions_per_lease_duration() [2/2] . . . 1652
8.163 dds::core::status::LivelinessChangedStatus Class Reference . . . 1652
8.163.1 Detailed Description . . . 1652
8.163.2 Member Function Documentation . . . 1653
8.163.2.1 alive_count() . . . 1653
8.163.2.2 not_alive_count() . . . 1653
8.163.2.3 alive_count_change() . . . 1653
8.163.2.4 not_alive_count_change() . . . 1654
8.163.2.5 last_publication_handle() . . . 1654
8.164 dds::core::policy::LivelinessKind_def Struct Reference . . . 1654
8.164.1 Detailed Description . . . 1654
8.164.2 Member Enumeration Documentation . . . 1654
8.164.2.1 type . . . 1654
8.165 dds::core::status::LivelinessLostStatus Class Reference . . . 1655
8.165.1 Detailed Description . . . 1655
8.165.2 Member Function Documentation . . . 1656
8.165.2.1 total_count() . . . 1656
8.165.2.2 total_count_change() . . . 1656

8.166 rti::core::xtypes::LoanedDynamicData Class Reference . . . 1656
8.166.1 Detailed Description . . . 1657
8.166.2 Constructor & Destructor Documentation . . . 1657
8.166.2.1 ∼LoanedDynamicData() . . . 1658
8.166.2.2 LoanedDynamicData() . . . 1658
8.166.3 Member Function Documentation . . . 1658
8.166.3.1 return_loan() . . . 1658
8.166.3.2 get() [1/2] . . . 1658
8.166.3.3 get() [2/2] . . . 1658
8.166.3.4 operator DynamicData &() . . . 1659
8.166.3.5 operator const DynamicData &() . . . 1659
8.166.3.6 operator=() . . . 1659
8.167 rti::sub::LoanedSample< T > Class Template Reference . . . 1659
8.167.1 Detailed Description . . . 1660
8.167.2 Member Typedef Documentation . . . 1661
8.167.2.1 DataType . . . 1661
8.167.2.2 InfoType . . . 1661
8.167.3 Member Function Documentation . . . 1661
8.167.3.1 data() . . . 1661
8.167.3.2 info() . . . 1662
8.167.3.3 operator const DataType &() . . . 1662
8.167.3.4 operator==() . . . 1662
8.167.4 Friends And Related Function Documentation . . . 1663
8.167.4.1 copy_to_sample() . . . 1663
8.167.4.2 operator<<() . . . 1663
8.168 dds::sub::LoanedSamples< T > Class Template Reference . . . 1663
8.168.1 Detailed Description . . . 1665
8.168.2 Member Typedef Documentation . . . 1665
8.168.2.1 iterator . . . 1665
8.168.3 Constructor & Destructor Documentation . . . 1666
8.168.3.1 LoanedSamples() [1/2] . . . 1666
8.168.3.2 ∼LoanedSamples() . . . 1666
8.168.3.3 LoanedSamples() [2/2] . . . 1666
8.168.4 Member Function Documentation . . . 1666
8.168.4.1 operator[]() . . . 1666
8.168.4.2 length() . . . 1667
8.168.4.3 return_loan() . . . 1667
8.168.4.4 return_loan_noexcept() . . . 1667
8.168.4.5 begin() [1/2] . . . 1668

8.168.4.6 end() [1/2] . . . 1668
8.168.4.7 begin() [2/2] . . . 1668
8.168.4.8 end() [2/2] . . . 1668
8.168.4.9 swap() . . . 1668
8.168.5 Friends And Related Function Documentation . . . 1669
8.168.5.1 move() . . . 1669
8.168.5.2 begin() [1/2] . . . 1669
8.168.5.3 begin() [2/2] . . . 1670
8.168.5.4 end() [1/2] . . . 1670
8.168.5.5 end() [2/2] . . . 1670
8.168.5.6 swap() . . . 1671
8.168.5.7 valid_data() [1/2] . . . 1671
8.168.5.8 valid_data() [2/2] . . . 1672
8.169 rti::core::Locator Class Reference . . . 1673
8.169.1 Detailed Description . . . 1673
8.169.2 Constructor & Destructor Documentation . . . 1674
8.169.2.1 Locator() . . . 1674
8.169.3 Member Function Documentation . . . 1674
8.169.3.1 kind() [1/2] . . . 1674
8.169.3.2 kind() [2/2] . . . 1674
8.169.3.3 port() [1/2] . . . 1675
8.169.3.4 port() [2/2] . . . 1675
8.169.3.5 address() [1/2] . . . 1675
8.169.3.6 address() [2/2] . . . 1675
8.169.3.7 Invalid() . . . 1675
8.169.4 Friends And Related Function Documentation . . . 1675
8.169.4.1 LocatorSeq . . . 1676
8.170 rti::core::policy::LocatorFilter Class Reference . . . 1676
8.170.1 Detailed Description . . . 1676
8.170.2 Member Typedef Documentation . . . 1677
8.170.2.1 Filter . . . 1677
8.170.2.2 FilterSeq . . . 1677
8.170.3 Constructor & Destructor Documentation . . . 1677
8.170.3.1 LocatorFilter() [1/2] . . . 1677
8.170.3.2 LocatorFilter() [2/2] . . . 1677
8.170.4 Member Function Documentation . . . 1677
8.170.4.1 locator_filters() [1/2] . . . 1678
8.170.4.2 locator_filters() [2/2] . . . 1678
8.170.4.3 filter_name() [1/2] . . . 1678

8.170.4.4 filter_name() [2/2] . . . 1678
8.171 rti::core::LocatorFilterElement Class Reference . . . 1678
8.171.1 Detailed Description . . . 1679
8.171.2 Constructor & Destructor Documentation . . . 1679
8.171.2.1 LocatorFilterElement() . . . 1679
8.171.3 Member Function Documentation . . . 1679
8.171.3.1 locators() [1/2] . . . 1680
8.171.3.2 locators() [2/2] . . . 1680
8.171.3.3 filter_expression() [1/2] . . . 1680
8.171.3.4 filter_expression() [2/2] . . . 1680
8.172 rti::core::LocatorKind_def Struct Reference . . . 1681
8.172.1 Detailed Description . . . 1681
8.172.2 Member Enumeration Documentation . . . 1681
8.172.2.1 type . . . 1681
8.173 rti::config::LogCategory_def Struct Reference . . . 1682
8.173.1 Detailed Description . . . 1682
8.173.2 Member Enumeration Documentation . . . 1682
8.173.2.1 type . . . 1682
8.174 rti::config::Logger Class Reference . . . 1683
8.174.1 Detailed Description . . . 1685
8.174.2 Member Function Documentation . . . 1685
8.174.2.1 instance() . . . 1685
8.174.2.2 verbosity() [1/2] . . . 1685
8.174.2.3 verbosity_by_category() [1/2] . . . 1686
8.174.2.4 verbosity() [2/2] . . . 1686
8.174.2.5 verbosity_by_category() [2/2] . . . 1686
8.174.2.6 output_file() [1/3] . . . 1686
8.174.2.7 output_file() [2/3] . . . 1687
8.174.2.8 output_file() [3/3] . . . 1687
8.174.2.9 output_file_set() . . . 1687
8.174.2.10 output_handler() . . . 1687
8.174.2.11 reset_output_handler() . . . 1688
8.174.2.12 print_format() [1/2] . . . 1688
8.174.2.13 print_format() [2/2] . . . 1688
8.174.2.14 print_format_by_log_level() [1/2] . . . 1689
8.174.2.15 print_format_by_log_level() [2/2] . . . 1689
8.174.2.16 emergency() [1/2] . . . 1689
8.174.2.17 alert() [1/2] . . . 1689
8.174.2.18 critical() [1/2] . . . 1689

8.174.2.19 error() [1/2] . . . 1690
8.174.2.20 warning() [1/2] . . . 1690
8.174.2.21 notice() [1/2] . . . 1690
8.174.2.22 informational() [1/2] . . . 1690
8.174.2.23 debug() [1/2] . . . 1690
8.174.2.24 emergency() [2/2] . . . 1691
8.174.2.25 alert() [2/2] . . . 1691
8.174.2.26 critical() [2/2] . . . 1691
8.174.2.27 error() [2/2] . . . 1691
8.174.2.28 warning() [2/2] . . . 1691
8.174.2.29 notice() [2/2] . . . 1692
8.174.2.30 informational() [2/2] . . . 1692
8.174.2.31 debug() [2/2] . . . 1692
8.175 rti::config::LogLevel_def Struct Reference . . . 1692
8.175.1 Detailed Description . . . 1693
8.175.2 Member Enumeration Documentation . . . 1693
8.175.2.1 type . . . 1693
8.176 rti::config::LogMessage Struct Reference . . . 1693
8.176.1 Detailed Description . . . 1694
8.176.2 Member Data Documentation . . . 1694
8.176.2.1 level . . . 1694
8.176.2.2 text . . . 1694
8.176.2.3 is_security_message . . . 1695
8.176.2.4 is_license_message . . . 1695
8.176.2.5 message_id . . . 1695
8.176.2.6 timestamp . . . 1695
8.176.2.7 facility . . . 1696
8.177 rti::core::LongDouble Class Reference . . . 1696
8.177.1 Detailed Description . . . 1696
8.177.2 Constructor & Destructor Documentation . . . 1696
8.177.2.1 LongDouble() . . . 1696
8.177.3 Member Function Documentation . . . 1697
8.177.3.1 operator[]() [1/2] . . . 1697
8.177.3.2 operator[]() [2/2] . . . 1697
8.178 dds::sub::DataReader< T >::ManipulatorSelector Class Reference . . . 1697
8.178.1 Detailed Description . . . 1698
8.178.2 Member Function Documentation . . . 1699
8.178.2.1 operator>>() [1/3] . . . 1699
8.178.2.2 operator>>() [2/3] . . . 1699

8.178.2.3 operator>>() [3/3] . . . 1699
8.179 rti::sub::ManipulatorSelector< T > Class Template Reference . . . 1700
8.179.1 Detailed Description . . . 1700
8.180 dds::core::xtypes::Member Class Reference . . . 1700
8.180.1 Detailed Description . . . 1702
8.180.2 Constructor & Destructor Documentation . . . 1702
8.180.2.1 Member() [1/2] . . . 1702
8.180.2.2 Member() [2/2] . . . 1702
8.180.3 Member Function Documentation . . . 1702
8.180.3.1 name() [1/3] . . . 1702
8.180.3.2 name() [2/3] . . . 1703
8.180.3.3 type() . . . 1703
8.180.3.4 has_id() . . . 1703
8.180.3.5 get_id() . . . 1703
8.180.3.6 is_pointer() . . . 1703
8.180.3.7 is_key() . . . 1703
8.180.3.8 is_optional() . . . 1704
8.180.3.9 is_must_understand() . . . 1704
8.180.3.10 is_bitset() . . . 1704
8.180.3.11 has_bitbound() . . . 1704
8.180.3.12 get_bitbound() . . . 1704
8.180.3.13 name() [3/3] . . . 1704
8.180.3.14 key() . . . 1705
8.180.3.15 optional() . . . 1705
8.180.3.16 must_understand() . . . 1705
8.180.3.17 id() . . . 1705
8.180.3.18 pointer() . . . 1706
8.180.4 Member Data Documentation . . . 1706
8.180.4.1 INVALID_ID . . . 1706
8.181 rti::core::policy::Monitoring Class Reference . . . 1706
8.181.1 Detailed Description . . . 1707
8.181.2 Constructor & Destructor Documentation . . . 1707
8.181.2.1 Monitoring() . . . 1707
8.181.3 Member Function Documentation . . . 1707
8.181.3.1 Enabled() . . . 1707
8.181.3.2 Disabled() . . . 1708
8.181.3.3 enable() [1/2] . . . 1708
8.181.3.4 enable() [2/2] . . . 1709
8.181.3.5 application_name() [1/3] . . . 1709

8.181.3.6 application_name() [2/3] . . . 1709
8.181.3.7 application_name() [3/3] . . . 1710
8.181.3.8 distribution_settings() [1/3] . . . 1710
8.181.3.9 distribution_settings() [2/3] . . . 1711
8.181.3.10 distribution_settings() [3/3] . . . 1711
8.181.3.11 telemetry_data() [1/3] . . . 1711
8.181.3.12 telemetry_data() [2/3] . . . 1711
8.181.3.13 telemetry_data() [3/3] . . . 1711
8.182 rti::core::MonitoringDedicatedParticipantSettings Class Reference . . . 1712
8.182.1 Detailed Description . . . 1712
8.182.2 Constructor & Destructor Documentation . . . 1712
8.182.2.1 MonitoringDedicatedParticipantSettings() . . . 1713
8.182.3 Member Function Documentation . . . 1713
8.182.3.1 enable() [1/2] . . . 1713
8.182.3.2 enable() [2/2] . . . 1713
8.182.3.3 domain_id() [1/2] . . . 1714
8.182.3.4 domain_id() [2/2] . . . 1714
8.182.3.5 participant_qos_profile_name() [1/3] . . . 1714
8.182.3.6 participant_qos_profile_name() [2/3] . . . 1715
8.182.3.7 participant_qos_profile_name() [3/3] . . . 1715
8.182.3.8 collector_initial_peers() [1/2] . . . 1715
8.182.3.9 collector_initial_peers() [2/2] . . . 1716
8.183 rti::core::MonitoringDistributionSettings Class Reference . . . 1716
8.183.1 Detailed Description . . . 1717
8.183.2 Constructor & Destructor Documentation . . . 1718
8.183.2.1 MonitoringDistributionSettings() . . . 1718
8.183.3 Member Function Documentation . . . 1718
8.183.3.1 publisher_qos_profile_name() [1/3] . . . 1718
8.183.3.2 publisher_qos_profile_name() [2/3] . . . 1718
8.183.3.3 publisher_qos_profile_name() [3/3] . . . 1719
8.183.3.4 dedicated_participant() [1/3] . . . 1719
8.183.3.5 dedicated_participant() [2/3] . . . 1719
8.183.3.6 dedicated_participant() [3/3] . . . 1720
8.183.3.7 event_settings() [1/3] . . . 1720
8.183.3.8 event_settings() [2/3] . . . 1720
8.183.3.9 event_settings() [3/3] . . . 1720
8.183.3.10 periodic_settings() [1/3] . . . 1721
8.183.3.11 periodic_settings() [2/3] . . . 1721
8.183.3.12 periodic_settings() [3/3] . . . 1721

8.183.3.13 logging_settings() [1/3] . . . 1721
8.183.3.14 logging_settings() [2/3] . . . 1722
8.183.3.15 logging_settings() [3/3] . . . 1722
8.184 rti::core::MonitoringEventDistributionSettings Class Reference . . . 1722
8.184.1 Detailed Description . . . 1723
8.184.2 Constructor & Destructor Documentation . . . 1723
8.184.2.1 MonitoringEventDistributionSettings() . . . 1723
8.184.3 Member Function Documentation . . . 1723
8.184.3.1 concurrency_level() [1/2] . . . 1724
8.184.3.2 concurrency_level() [2/2] . . . 1724
8.184.3.3 datawriter_qos_profile_name() [1/3] . . . 1724
8.184.3.4 datawriter_qos_profile_name() [2/3] . . . 1725
8.184.3.5 datawriter_qos_profile_name() [3/3] . . . 1726
8.184.3.6 thread() [1/3] . . . 1726
8.184.3.7 thread() [2/3] . . . 1726
8.184.3.8 thread() [3/3] . . . 1726
8.184.3.9 publication_period() [1/2] . . . 1727
8.184.3.10 publication_period() [2/2] . . . 1727
8.185 rti::core::MonitoringLoggingDistributionSettings Class Reference . . . 1727
8.185.1 Detailed Description . . . 1728
8.185.2 Constructor & Destructor Documentation . . . 1728
8.185.2.1 MonitoringLoggingDistributionSettings() . . . 1729
8.185.3 Member Function Documentation . . . 1729
8.185.3.1 concurrency_level() [1/2] . . . 1729
8.185.3.2 concurrency_level() [2/2] . . . 1729
8.185.3.3 max_historical_logs() [1/2] . . . 1730
8.185.3.4 max_historical_logs() [2/2] . . . 1730
8.185.3.5 datawriter_qos_profile_name() [1/3] . . . 1730
8.185.3.6 datawriter_qos_profile_name() [2/3] . . . 1731
8.185.3.7 datawriter_qos_profile_name() [3/3] . . . 1731
8.185.3.8 thread() [1/3] . . . 1732
8.185.3.9 thread() [2/3] . . . 1732
8.185.3.10 thread() [3/3] . . . 1732
8.185.3.11 publication_period() [1/2] . . . 1732
8.185.3.12 publication_period() [2/2] . . . 1733
8.186 rti::core::MonitoringLoggingForwardingSettings Class Reference . . . 1733
8.186.1 Detailed Description . . . 1733
8.186.2 Member Function Documentation . . . 1734
8.186.2.1 middleware_forwarding_level() [1/2] . . . 1734

8.186.2.2 middleware_forwarding_level() [2/2] . . . 1734
8.186.2.3 security_event_forwarding_level() [1/2] . . . 1734
8.186.2.4 security_event_forwarding_level() [2/2] . . . 1735
8.186.2.5 service_forwarding_level() [1/2] . . . 1735
8.186.2.6 service_forwarding_level() [2/2] . . . 1736
8.186.2.7 user_forwarding_level() [1/2] . . . 1736
8.186.2.8 user_forwarding_level() [2/2] . . . 1736
8.187 rti::core::MonitoringMetricSelection Class Reference . . . 1736
8.187.1 Detailed Description . . . 1737
8.187.2 Constructor & Destructor Documentation . . . 1737
8.187.2.1 MonitoringMetricSelection() . . . 1737
8.187.3 Member Function Documentation . . . 1737
8.187.3.1 resource_selection() [1/2] . . . 1737
8.187.3.2 resource_selection() [2/2] . . . 1738
8.187.3.3 enabled_metrics_selection() [1/2] . . . 1739
8.187.3.4 enabled_metrics_selection() [2/2] . . . 1739
8.187.3.5 disabled_metrics_selection() [1/2] . . . 1740
8.187.3.6 disabled_metrics_selection() [2/2] . . . 1740
8.188 rti::core::MonitoringPeriodicDistributionSettings Class Reference . . . 1740
8.188.1 Detailed Description . . . 1741
8.188.2 Constructor & Destructor Documentation . . . 1741
8.188.2.1 MonitoringPeriodicDistributionSettings() . . . 1741
8.188.3 Member Function Documentation . . . 1742
8.188.3.1 datawriter_qos_profile_name() [1/3] . . . 1742
8.188.3.2 datawriter_qos_profile_name() [2/3] . . . 1742
8.188.3.3 datawriter_qos_profile_name() [3/3] . . . 1743
8.188.3.4 thread() [1/3] . . . 1743
8.188.3.5 thread() [2/3] . . . 1743
8.188.3.6 thread() [3/3] . . . 1744
8.188.3.7 polling_period() [1/2] . . . 1744
8.188.3.8 polling_period() [2/2] . . . 1744
8.189 rti::core::MonitoringTelemetryData Class Reference . . . 1744
8.189.1 Detailed Description . . . 1745
8.189.2 Member Function Documentation . . . 1745
8.189.2.1 metrics() [1/2] . . . 1745
8.189.2.2 metrics() [2/2] . . . 1746
8.189.2.3 logs() [1/3] . . . 1746
8.189.2.4 logs() [2/3] . . . 1746
8.189.2.5 logs() [3/3] . . . 1747

8.190 rti::core::policy::MulticastMapping Class Reference 1747
8.190.1 Detailed Description 1747
8.190.2 Member Function Documentation 1748
8.190.2.1 addresses() [1/2] 1748
8.190.2.2 addresses() [2/2] 1748
8.190.2.3 topic_expression() [1/2] 1749
8.190.2.4 topic_expression() [2/2] 1749
8.190.2.5 mapping_function() [1/2] 1749
8.190.2.6 mapping_function() [2/2] 1749
8.191 rti::core::policy::MultiChannel Class Reference 1749
8.191.1 Detailed Description 1750
8.191.2 Usage 1750
8.191.3 Constructor & Destructor Documentation 1751
8.191.3.1 MultiChannel() [1/2] 1751
8.191.3.2 MultiChannel() [2/2] 1751
8.191.4 Member Function Documentation 1751
8.191.4.1 channels() [1/2] 1752
8.191.4.2 channels() [2/2] 1752
8.191.4.3 filter_name() [1/2] 1752
8.191.4.4 filter_name() [2/2] 1753
8.192 rti::flat::MutableArrayBuilder< ElementBuilder, N > Class Template Reference 1753
8.192.1 Detailed Description 1753
8.192.2 Member Typedef Documentation 1754
8.192.2.1 Offset 1754
8.192.3 Member Function Documentation 1754
8.192.3.1 build_next() 1754
8.192.3.2 finish() 1755
8.193 rti::flat::MutableArrayOffset< ElementOffset, N > Class Template Reference 1755
8.193.1 Detailed Description 1755
8.193.2 Member Function Documentation 1756
8.193.2.1 get_element() 1756
8.194 rti::flat::MutableOffset Class Reference 1756
8.194.1 Detailed Description 1757
8.195 rti::flat::MutableSequenceBuilder< ElementBuilder > Class Template Reference 1757
8.195.1 Detailed Description 1757
8.195.2 Member Typedef Documentation 1758
8.195.2.1 Offset 1758
8.195.3 Member Function Documentation 1758
8.195.3.1 build_next() 1758

8.195.3.2 finish() . . . 1759
8.196 MyFlatFinalOffset Class Reference . . . 1759
8.196.1 Detailed Description . . . 1760
8.196.2 Member Typedef Documentation . . . 1760
8.196.2.1 ConstOffset . . . 1760
8.196.3 Constructor & Destructor Documentation . . . 1761
8.196.3.1 MyFlatFinalOffset() . . . 1761
8.196.4 Member Function Documentation . . . 1761
8.196.4.1 my_primitive() [1/2] . . . 1761
8.196.4.2 my_complex() [1/2] . . . 1761
8.196.4.3 my_primitive_array() [1/2] . . . 1761
8.196.4.4 my_complex_array() [1/2] . . . 1762
8.196.4.5 my_primitive() [2/2] . . . 1762
8.196.4.6 my_complex() [2/2] . . . 1762
8.196.4.7 my_primitive_array() [2/2] . . . 1762
8.196.4.8 my_complex_array() [2/2] . . . 1762
8.197 MyFlatMutableBuilder Class Reference . . . 1763
8.197.1 Detailed Description . . . 1764
8.197.1.1 Adding fixed-size members . . . 1764
8.197.1.2 Building variable-size members . . . 1764
8.197.1.3 Choosing which members are included . . . 1765
8.197.2 Member Typedef Documentation . . . 1765
8.197.2.1 Offset . . . 1765
8.197.3 Constructor & Destructor Documentation . . . 1765
8.197.3.1 MyFlatMutableBuilder() [1/2] . . . 1765
8.197.3.2 MyFlatMutableBuilder() [2/2] . . . 1766
8.197.4 Member Function Documentation . . . 1766
8.197.4.1 finish() . . . 1766
8.197.4.2 finish_sample() . . . 1767
8.197.4.3 add_my_primitive() . . . 1767
8.197.4.4 add_my_optional_primitive() . . . 1767
8.197.4.5 add_my_primitive_array() . . . 1768
8.197.4.6 build_my_primitive_seq() . . . 1768
8.197.4.7 add_my_final() . . . 1768
8.197.4.8 add_my_final_array() . . . 1768
8.197.4.9 build_my_final_seq() . . . 1769
8.197.4.10 build_my_mutable() . . . 1769
8.197.4.11 build_my_mutable_array() . . . 1769
8.197.4.12 build_my_mutable_seq() . . . 1769

8.197.4.13 build_my_string() . . . 1769
8.197.4.14 build_my_string_seq() . . . 1770
8.198 MyFlatMutableOffset Class Reference . . . 1770
8.198.1 Detailed Description . . . 1771
8.198.2 Member Typedef Documentation . . . 1771
8.198.2.1 ConstOffset . . . 1772
8.198.3 Constructor & Destructor Documentation . . . 1772
8.198.3.1 MyFlatMutableOffset() . . . 1772
8.198.4 Member Function Documentation . . . 1772
8.198.4.1 my_primitive() [1/2] . . . 1772
8.198.4.2 my_optional_primitive() . . . 1773
8.198.4.3 my_primitive_array() [1/2] . . . 1773
8.198.4.4 my_primitive_seq() [1/2] . . . 1773
8.198.4.5 my_final() [1/2] . . . 1773
8.198.4.6 my_final_array() [1/2] . . . 1773
8.198.4.7 my_final_seq() [1/2] . . . 1774
8.198.4.8 my_mutable() [1/2] . . . 1774
8.198.4.9 my_mutable_array() [1/2] . . . 1774
8.198.4.10 my_mutable_seq() [1/2] . . . 1774
8.198.4.11 my_string() [1/2] . . . 1774
8.198.4.12 my_string_seq() [1/2] . . . 1775
8.198.4.13 my_primitive() [2/2] . . . 1775
8.198.4.14 my_primitive_array() [2/2] . . . 1775
8.198.4.15 my_primitive_seq() [2/2] . . . 1775
8.198.4.16 my_final() [2/2] . . . 1775
8.198.4.17 my_final_array() [2/2] . . . 1776
8.198.4.18 my_final_seq() [2/2] . . . 1776
8.198.4.19 my_mutable() [2/2] . . . 1776
8.198.4.20 my_mutable_array() [2/2] . . . 1776
8.198.4.21 my_mutable_seq() [2/2] . . . 1776
8.198.4.22 my_string() [2/2] . . . 1777
8.198.4.23 my_string_seq() [2/2] . . . 1777
8.199 MyFlatUnionBuilder Class Reference . . . 1777
8.199.1 Detailed Description . . . 1778
8.199.2 Member Typedef Documentation . . . 1778
8.199.2.1 Offset . . . 1778
8.199.3 Constructor & Destructor Documentation . . . 1778
8.199.3.1 MyFlatUnionBuilder() . . . 1778
8.199.4 Member Function Documentation . . . 1779

8.199.4.1 finish() . . . 1779
8.199.4.2 finish_sample() . . . 1779
8.199.4.3 add_my_primitive() . . . 1779
8.199.4.4 build_my_mutable() . . . 1779
8.199.4.5 add_my_final() . . . 1780
8.200 MyFlatUnionOffset Class Reference . . . 1780
8.200.1 Detailed Description . . . 1781
8.200.2 Member Typedef Documentation . . . 1781
8.200.2.1 ConstOffset . . . 1781
8.200.3 Constructor & Destructor Documentation . . . 1781
8.200.3.1 MyFlatUnionOffset() . . . 1782
8.200.4 Member Function Documentation . . . 1782
8.200.4.1 _d() . . . 1782
8.200.4.2 my_primitive() [1/2] . . . 1782
8.200.4.3 my_mutable() [1/2] . . . 1783
8.200.4.4 my_final() [1/2] . . . 1783
8.200.4.5 my_primitive() [2/2] . . . 1783
8.200.4.6 my_mutable() [2/2] . . . 1784
8.200.4.7 my_final() [2/2] . . . 1784
8.201 NDDS_Transport_Address_t Struct Reference . . . 1784
8.201.1 Detailed Description . . . 1784
8.201.2 Member Data Documentation . . . 1785
8.201.2.1 network_ordered_value . . . 1785
8.202 NDDS_Transport_Interface_t Struct Reference . . . 1785
8.202.1 Detailed Description . . . 1785
8.202.2 Member Data Documentation . . . 1785
8.202.2.1 transport_classid . . . 1785
8.202.2.2 address . . . 1786
8.202.2.3 status . . . 1786
8.202.2.4 rank . . . 1786
8.203 NDDS_Transport_Property_t Struct Reference . . . 1786
8.203.1 Detailed Description . . . 1787
8.203.2 Member Data Documentation . . . 1787
8.203.2.1 classid . . . 1788
8.203.2.2 address_bit_count . . . 1788
8.203.2.3 properties_bitmap . . . 1789
8.203.2.4 gather_send_buffer_count_max . . . 1789
8.203.2.5 message_size_max . . . 1789
8.203.2.6 allow_interfaces_list . . . 1790

8.203.2.7 allow_interfaces_list_length . . . 1790
8.203.2.8 deny_interfaces_list . . . 1791
8.203.2.9 deny_interfaces_list_length . . . 1791
8.203.2.10 allow_multicast_interfaces_list . . . 1792
8.203.2.11 allow_multicast_interfaces_list_length . . . 1792
8.203.2.12 deny_multicast_interfaces_list . . . 1792
8.203.2.13 deny_multicast_interfaces_list_length . . . 1793
8.203.2.14 transport_uuid . . . 1793
8.203.2.15 thread_name_prefix . . . 1793
8.203.2.16 max_interface_count . . . 1794
8.204 NDDS_Transport_Shmem_Property_t Struct Reference . . . 1794
8.204.1 Detailed Description . . . 1795
8.204.2 Member Data Documentation . . . 1795
8.204.2.1 parent . . . 1795
8.204.2.2 received_message_count_max . . . 1795
8.204.2.3 receive_buffer_size . . . 1796
8.204.2.4 enable_udp_debugging . . . 1796
8.204.2.5 udp_debugging_address . . . 1797
8.204.2.6 udp_debugging_port . . . 1797
8.205 NDDS_Transport_UDP_WAN_CommPortsMappingInfo Struct Reference . . . 1797
8.205.1 Detailed Description . . . 1797
8.205.2 Member Data Documentation . . . 1797
8.205.2.1 rtps_port . . . 1798
8.205.2.2 host_port . . . 1798
8.205.2.3 public_port . . . 1798
8.206 NDDS_Transport_UDPv4_Property_t Struct Reference . . . 1798
8.206.1 Detailed Description . . . 1799
8.206.2 Member Data Documentation . . . 1800
8.206.2.1 parent . . . 1800
8.206.2.2 send_socket_buffer_size . . . 1800
8.206.2.3 recv_socket_buffer_size . . . 1801
8.206.2.4 unicast_enabled . . . 1801
8.206.2.5 multicast_enabled . . . 1801
8.206.2.6 multicast_ttl . . . 1802
8.206.2.7 multicast_loopback_disabled . . . 1802
8.206.2.8 ignore_loopback_interface . . . 1802
8.206.2.9 ignore_nonup_interfaces . . . 1803
8.206.2.10 ignore_nonrunning_interfaces . . . 1803
8.206.2.11 no_zero_copy . . . 1804

8.206.2.12 send_blocking . . . 1804
8.206.2.13 use_checksum . . . 1804
8.206.2.14 transport_priority_mask . . . 1805
8.206.2.15 transport_priority_mapping_low . . . 1805
8.206.2.16 transport_priority_mapping_high . . . 1805
8.206.2.17 send_ping . . . 1806
8.206.2.18 force_interface_poll_detection . . . 1806
8.206.2.19 interface_poll_period . . . 1806
8.206.2.20 reuse_multicast_receive_resource . . . 1806
8.206.2.21 protocol_overhead_max . . . 1807
8.206.2.22 disable_interface_tracking . . . 1807
8.206.2.23 join_multicast_group_timeout . . . 1807
8.206.2.24 public_address . . . 1808
8.206.2.25 port_offset . . . 1808
8.207 NDDS_Transport_UDPv4_WAN_Property_t Struct Reference . . . 1809
8.207.1 Detailed Description . . . 1810
8.207.2 Member Data Documentation . . . 1810
8.207.2.1 parent . . . 1810
8.207.2.2 send_socket_buffer_size . . . 1810
8.207.2.3 recv_socket_buffer_size . . . 1811
8.207.2.4 ignore_loopback_interface . . . 1811
8.207.2.5 ignore_nonup_interfaces . . . 1812
8.207.2.6 ignore_nonrunning_interfaces . . . 1812
8.207.2.7 no_zero_copy . . . 1813
8.207.2.8 send_blocking . . . 1813
8.207.2.9 use_checksum . . . 1813
8.207.2.10 transport_priority_mask . . . 1814
8.207.2.11 transport_priority_mapping_low . . . 1814
8.207.2.12 transport_priority_mapping_high . . . 1814
8.207.2.13 send_ping . . . 1815
8.207.2.14 force_interface_poll_detection . . . 1815
8.207.2.15 interface_poll_period . . . 1815
8.207.2.16 protocol_overhead_max . . . 1815
8.207.2.17 disable_interface_tracking . . . 1816
8.207.2.18 public_address . . . 1816
8.207.2.19 comm_ports_list . . . 1817
8.207.2.20 comm_ports_list_length . . . 1818
8.207.2.21 port_offset . . . 1818
8.207.2.22 binding_ping_period . . . 1818

8.208 NDDS_Transport_UDPv6_Property_t Struct Reference . . . 1819
8.208.1 Detailed Description . . . 1820
8.208.2 Member Data Documentation . . . 1820
8.208.2.1 parent . . . 1820
8.208.2.2 send_socket_buffer_size . . . 1821
8.208.2.3 recv_socket_buffer_size . . . 1821
8.208.2.4 unicast_enabled . . . 1822
8.208.2.5 multicast_enabled . . . 1822
8.208.2.6 multicast_ttl . . . 1822
8.208.2.7 multicast_loopback_disabled . . . 1822
8.208.2.8 ignore_loopback_interface . . . 1823
8.208.2.9 ignore_nonrunning_interfaces . . . 1823
8.208.2.10 no_zero_copy . . . 1824
8.208.2.11 send_blocking . . . 1824
8.208.2.12 enable_v4mapped . . . 1824
8.208.2.13 transport_priority_mask . . . 1825
8.208.2.14 transport_priority_mapping_low . . . 1825
8.208.2.15 transport_priority_mapping_high . . . 1825
8.208.2.16 send_ping . . . 1826
8.208.2.17 force_interface_poll_detection . . . 1826
8.208.2.18 interface_poll_period . . . 1826
8.208.2.19 reuse_multicast_receive_resource . . . 1826
8.208.2.20 protocol_overhead_max . . . 1827
8.208.2.21 disable_interface_tracking . . . 1827
8.208.2.22 join_multicast_group_timeout . . . 1827
8.208.2.23 public_address . . . 1828
8.208.2.24 port_offset . . . 1828
8.209 NDDS_Transport_UUID Struct Reference . . . 1829
8.209.1 Detailed Description . . . 1829
8.210 rti::util::network_capture::NetworkCaptureParams Class Reference . . . 1829
8.210.1 Detailed Description . . . 1830
8.210.2 Member Function Documentation . . . 1830
8.210.2.1 transports() [1/2] . . . 1830
8.210.2.2 transports() [2/2] . . . 1830
8.210.2.3 dropped_content() [1/2] . . . 1831
8.210.2.4 dropped_content() [2/2] . . . 1831
8.210.2.5 traffic() [1/2] . . . 1831
8.210.2.6 traffic() [2/2] . . . 1832
8.210.2.7 parse_encrypted_content() [1/2] . . . 1832

8.210.2.8 parse_encrypted_content() [2/2] . . . 1832
8.210.2.9 checkpoint_thread_settings() [1/2] . . . 1833
8.210.2.10 checkpoint_thread_settings() [2/2] . . . 1833
8.210.2.11 frame_queue_size() [1/2] . . . 1834
8.210.2.12 frame_queue_size() [2/2] . . . 1834
8.211 rti::topic::no_compile_data_t Struct Reference . . . 1835
8.211.1 Detailed Description . . . 1835
8.212 rti::queuing::NoMatchingQueueException Class Reference . . . 1835
8.212.1 Detailed Description . . . 1836
8.212.2 Member Function Documentation . . . 1836
8.212.2.1 what() . . . 1836
8.213 rti::core::cond::NoOpAsyncWaitSetListener Class Reference . . . 1836
8.213.1 Detailed Description . . . 1837
8.213.2 Member Function Documentation . . . 1837
8.213.2.1 on_thread_spawned() . . . 1837
8.213.2.2 on_thread_deleted() . . . 1837
8.213.2.3 on_wait_timeout() . . . 1837
8.214 dds::sub::NoOpDataReaderListener< T > Class Template Reference . . . 1838
8.214.1 Detailed Description . . . 1838
8.214.2 Member Function Documentation . . . 1839
8.214.2.1 on_requested_deadline_missed() . . . 1839
8.214.2.2 on_requested_incompatible_qos() . . . 1839
8.214.2.3 on_sample_rejected() . . . 1839
8.214.2.4 on_liveliness_changed() . . . 1840
8.214.2.5 on_data_available() . . . 1840
8.214.2.6 on_subscription_matched() . . . 1840
8.214.2.7 on_sample_lost() . . . 1840
8.215 dds::pub::NoOpDataWriterListener< T > Class Template Reference . . . 1841
8.215.1 Detailed Description . . . 1842
8.215.2 Member Function Documentation . . . 1842
8.215.2.1 on_offered_deadline_missed() . . . 1842
8.215.2.2 on_offered_incompatible_qos() . . . 1842
8.215.2.3 on_liveliness_lost() . . . 1842
8.215.2.4 on_publication_matched() . . . 1843
8.215.2.5 on_reliable_writer_cache_changed() . . . 1843
8.215.2.6 on_reliable_reader_activity_changed() . . . 1843
8.215.2.7 on_instance_replaced() . . . 1843
8.215.2.8 on_application_acknowledgment() . . . 1844
8.215.2.9 on_service_request_accepted() . . . 1844

8.215.2.10 on_destination_unreachable() . . . 1844
8.215.2.11 on_data_request() . . . 1844
8.215.2.12 on_data_return() . . . 1845
8.215.2.13 on_sample_removed() . . . 1845
8.216 dds::domain::NoOpDomainParticipantListener Class Reference . . . 1845
8.216.1 Detailed Description . . . 1847
8.216.2 Member Function Documentation . . . 1847
8.216.2.1 on_offered_deadline_missed() . . . 1847
8.216.2.2 on_offered_incompatible_qos() . . . 1847
8.216.2.3 on_liveliness_lost() . . . 1848
8.216.2.4 on_publication_matched() . . . 1848
8.216.2.5 on_reliable_writer_cache_changed() . . . 1848
8.216.2.6 on_reliable_reader_activity_changed() . . . 1848
8.216.2.7 on_sample_removed() . . . 1849
8.216.2.8 on_instance_replaced() . . . 1849
8.216.2.9 on_application_acknowledgment() . . . 1849
8.216.2.10 on_service_request_accepted() . . . 1849
8.216.2.11 on_invalid_local_status_advance_notice() . . . 1850
8.216.2.12 on_data_on_readers() . . . 1850
8.216.2.13 on_requested_deadline_missed() . . . 1850
8.216.2.14 on_requested_incompatible_qos() . . . 1850
8.216.2.15 on_sample_rejected() . . . 1851
8.216.2.16 on_liveliness_changed() . . . 1851
8.216.2.17 on_data_available() . . . 1851
8.216.2.18 on_subscription_matched() . . . 1851
8.216.2.19 on_sample_lost() . . . 1852
8.216.2.20 on_inconsistent_topic() . . . 1852
8.217 dds::pub::NoOpPublisherListener Class Reference . . . 1852
8.217.1 Detailed Description . . . 1853
8.217.2 Member Function Documentation . . . 1853
8.217.2.1 on_offered_deadline_missed() . . . 1854
8.217.2.2 on_offered_incompatible_qos() . . . 1854
8.217.2.3 on_liveliness_lost() . . . 1854
8.217.2.4 on_publication_matched() . . . 1854
8.217.2.5 on_reliable_writer_cache_changed() . . . 1855
8.217.2.6 on_reliable_reader_activity_changed() . . . 1855
8.217.2.7 on_sample_removed() . . . 1855
8.217.2.8 on_instance_replaced() . . . 1855
8.217.2.9 on_application_acknowledgment() . . . 1856

8.217.2.10 on_service_request_accepted() . . . 1856
8.218 rti::queuing::NoOpQueueConsumerListener< T > Class Template Reference . . . 1856
8.218.1 Detailed Description . . . 1857
8.218.2 Member Function Documentation . . . 1857
8.218.2.1 on_sample_available() . . . 1857
8.218.2.2 on_shared_reader_queue_matched() . . . 1857
8.219 rti::queuing::NoOpQueueProducerListener< T > Class Template Reference . . . 1858
8.219.1 Detailed Description . . . 1858
8.219.2 Member Function Documentation . . . 1858
8.219.2.1 on_sample_acknowledged() . . . 1858
8.219.2.2 on_shared_reader_queue_matched() . . . 1859
8.220 rti::queuing::NoOpQueueReplierListener< TReq, TRep > Class Template Reference . . . 1859
8.220.1 Detailed Description . . . 1860
8.220.2 Member Function Documentation . . . 1860
8.220.2.1 on_reply_acknowledged() . . . 1860
8.220.2.2 on_request_available() . . . 1860
8.220.2.3 on_request_shared_reader_queue_matched() . . . 1861
8.220.2.4 on_reply_shared_reader_queue_matched() . . . 1861
8.221 rti::queuing::NoOpQueueRequesterListener< TReq, TRep > Class Template Reference . . . 1861
8.221.1 Detailed Description . . . 1862
8.221.2 Member Function Documentation . . . 1862
8.221.2.1 on_request_acknowledged() . . . 1862
8.221.2.2 on_reply_available() . . . 1863
8.221.2.3 on_request_shared_reader_queue_matched() . . . 1863
8.221.2.4 on_reply_shared_reader_queue_matched() . . . 1863
8.222 dds::sub::NoOpSubscriberListener Class Reference . . . 1864
8.222.1 Detailed Description . . . 1864
8.222.2 Member Function Documentation . . . 1865
8.222.2.1 on_data_on_readers() . . . 1865
8.222.2.2 on_requested_deadline_missed() . . . 1865
8.222.2.3 on_requested_incompatible_qos() . . . 1865
8.222.2.4 on_sample_rejected() . . . 1866
8.222.2.5 on_liveliness_changed() . . . 1866
8.222.2.6 on_data_available() . . . 1866
8.222.2.7 on_subscription_matched() . . . 1866
8.222.2.8 on_sample_lost() . . . 1867
8.223 dds::topic::NoOpTopicListener< T > Class Template Reference . . . 1867
8.223.1 Detailed Description . . . 1867
8.223.2 Member Function Documentation . . . 1867

8.223.2.1 on_inconsistent_topic() . . . 1868
8.224 dds::core::NotAllowedBySecurityError Class Reference . . . 1868
8.224.1 Detailed Description . . . 1868
8.224.2 Member Function Documentation . . . 1868
8.224.2.1 what() . . . 1869
8.225 dds::core::NotEnabledError Class Reference . . . 1869
8.225.1 Detailed Description . . . 1869
8.225.2 Member Function Documentation . . . 1869
8.225.2.1 what() . . . 1870
8.226 dds::core::NullReferenceError Class Reference . . . 1870
8.226.1 Detailed Description . . . 1870
8.226.2 Member Function Documentation . . . 1870
8.226.2.1 what() . . . 1871
8.227 rti::core::xtypes::octet_tag_t Struct Reference . . . 1871
8.227.1 Detailed Description . . . 1871
8.228 dds::core::status::OfferedDeadlineMissedStatus Class Reference . . . 1871
8.228.1 Detailed Description . . . 1872
8.228.2 Member Function Documentation . . . 1872
8.228.2.1 total_count() . . . 1872
8.228.2.2 total_count_change() . . . 1872
8.228.2.3 last_instance_handle() . . . 1872
8.229 dds::core::status::OfferedIncompatibleQosStatus Class Reference . . . 1873
8.229.1 Detailed Description . . . 1873
8.229.2 Member Function Documentation . . . 1873
8.229.2.1 total_count() . . . 1874
8.229.2.2 total_count_change() . . . 1874
8.229.2.3 last_policy_id() . . . 1874
8.229.2.4 policies() . . . 1874
8.230 rti::flat::OffsetBase Class Reference . . . 1875
8.230.1 Detailed Description . . . 1876
8.230.2 Member Function Documentation . . . 1876
8.230.2.1 is_null() . . . 1876
8.230.2.2 is_cpp_compatible() . . . 1877
8.230.2.3 get_buffer() . . . 1877
8.230.2.4 get_buffer_size() . . . 1877
8.230.3 Friends And Related Function Documentation . . . 1878
8.230.3.1 operator< . . . 1878
8.230.3.2 operator> . . . 1878
8.230.3.3 operator<= . . . 1878

8.230.3.4 operator>= . . . 1879
8.230.3.5 operator== . . . 1879
8.230.3.6 operator"!= . . . 1879
8.231 dds::core::optional< T > Class Template Reference . . . 1879
8.231.1 Detailed Description . . . 1881
8.231.2 Constructor & Destructor Documentation . . . 1882
8.231.2.1 optional() [1/5] . . . 1882
8.231.2.2 optional() [2/5] . . . 1882
8.231.2.3 optional() [3/5] . . . 1883
8.231.2.4 optional() [4/5] . . . 1883
8.231.2.5 optional() [5/5] . . . 1883
8.231.2.6 ~optional() . . . 1884
8.231.3 Member Function Documentation . . . 1884
8.231.3.1 set() [1/2] . . . 1884
8.231.3.2 set() [2/2] . . . 1884
8.231.3.3 is_set() . . . 1884
8.231.3.4 has_value() . . . 1885
8.231.3.5 operator bool() . . . 1885
8.231.3.6 reset() . . . 1885
8.231.3.7 value() [1/2] . . . 1885
8.231.3.8 value() [2/2] . . . 1886
8.231.3.9 get() [1/2] . . . 1886
8.231.3.10 get() [2/2] . . . 1886
8.231.3.11 get_ptr() [1/2] . . . 1887
8.231.3.12 get_ptr() [2/2] . . . 1887
8.231.3.13 operator*() [1/2] . . . 1887
8.231.3.14 operator*() [2/2] . . . 1887
8.231.3.15 operator->() [1/2] . . . 1887
8.231.3.16 operator->() [2/2] . . . 1888
8.231.3.17 operator=() [1/4] . . . 1888
8.231.3.18 operator=() [2/4] . . . 1888
8.231.3.19 operator=() [3/4] . . . 1888
8.231.3.20 operator=() [4/4] . . . 1889
8.231.4 Friends And Related Function Documentation . . . 1889
8.231.4.1 swap . . . 1889
8.231.4.2 operator==() [1/3] . . . 1890
8.231.4.3 operator"!=() [1/3] . . . 1890
8.231.4.4 operator==() [2/3] . . . 1890
8.231.4.5 operator==() [3/3] . . . 1891

8.231.4.6 operator"!=() [2/3] . . . 1891
8.231.4.7 operator"!=() [3/3] . . . 1891
8.231.4.8 operator<<() . . . 1892
8.232 rti::core::optional_value< T > Class Template Reference . . . 1892
8.232.1 Detailed Description . . . 1893
8.232.2 Constructor & Destructor Documentation . . . 1893
8.232.2.1 optional_value() [1/4] . . . 1893
8.232.2.2 optional_value() [2/4] . . . 1893
8.232.2.3 optional_value() [3/4] . . . 1893
8.232.2.4 optional_value() [4/4] . . . 1894
8.232.3 Member Function Documentation . . . 1894
8.232.3.1 is_set() . . . 1894
8.232.3.2 has_value() . . . 1894
8.232.3.3 operator bool() . . . 1895
8.232.3.4 reset() . . . 1895
8.232.3.5 value() [1/2] . . . 1895
8.232.3.6 value() [2/2] . . . 1895
8.232.3.7 get() [1/2] . . . 1896
8.232.3.8 get() [2/2] . . . 1896
8.232.3.9 operator*() [1/2] . . . 1897
8.232.3.10 operator*() [2/2] . . . 1897
8.232.3.11 operator->() [1/2] . . . 1897
8.232.3.12 operator->() [2/2] . . . 1897
8.232.4 Friends And Related Function Documentation . . . 1898
8.232.4.1 swap . . . 1898
8.233 OsapiFile Class Reference . . . 1898
8.233.1 Detailed Description . . . 1898
8.234 dds::core::OutOfResourcesError Class Reference . . . 1898
8.234.1 Detailed Description . . . 1899
8.234.2 Member Function Documentation . . . 1899
8.234.2.1 what() . . . 1899
8.235 dds::core::policy::Ownership Class Reference . . . 1899
8.235.1 Detailed Description . . . 1900
8.235.2 Usage . . . 1900
8.235.2.1 SHARED ownership . . . 1900
8.235.2.2 EXCLUSIVE ownership . . . 1901
8.235.3 Compatibility . . . 1901
8.235.4 Relationship between registration, liveliness and ownership . . . 1902
8.235.4.1 Ownership Resolution on Redundant Systems . . . 1902

8.235.4.2 Detection of Loss in Topological Connectivity . . . 1903
8.235.4.3 Semantic Difference between unregister_instance and dispose . . . 1904
8.235.5 Constructor & Destructor Documentation . . . 1904
8.235.5.1 Ownership() [1/2] . . . 1904
8.235.5.2 Ownership() [2/2] . . . 1904
8.235.6 Member Function Documentation . . . 1904
8.235.6.1 kind() [1/2] . . . 1905
8.235.6.2 kind() [2/2] . . . 1905
8.235.6.3 Exclusive() . . . 1905
8.235.6.4 Shared() . . . 1905
8.236 dds::core::policy::OwnershipKind_def Struct Reference . . . 1905
8.236.1 Detailed Description . . . 1906
8.236.2 Member Enumeration Documentation . . . 1906
8.236.2.1 type . . . 1906
8.237 dds::core::policy::OwnershipStrength Class Reference . . . 1906
8.237.1 Detailed Description . . . 1907
8.237.2 Constructor & Destructor Documentation . . . 1907
8.237.2.1 OwnershipStrength() [1/2] . . . 1907
8.237.2.2 OwnershipStrength() [2/2] . . . 1908
8.237.3 Member Function Documentation . . . 1908
8.237.3.1 value() [1/2] . . . 1908
8.237.3.2 value() [2/2] . . . 1908
8.238 dds::topic::ParticipantBuiltinTopicData Class Reference . . . 1908
8.238.1 Detailed Description . . . 1909
8.238.2 Member Function Documentation . . . 1910
8.238.2.1 TParticipantBuiltinTopicData() . . . 1910
8.238.2.2 key() . . . 1910
8.238.2.3 user_data() . . . 1910
8.238.2.4 property() . . . 1910
8.238.2.5 rtps_protocol_version() . . . 1911
8.238.2.6 rtps_vendor_id() . . . 1911
8.238.2.7 dds_builtin_endpoints() . . . 1911
8.238.2.8 available_builtin_endpoints_ext() . . . 1912
8.238.2.9 default_unicast_locators() . . . 1912
8.238.2.10 partition() . . . 1913
8.238.2.11 trust_protection_info() . . . 1913
8.238.2.12 trust_algorithm_info() . . . 1913
8.238.2.13 product_version() . . . 1914
8.238.2.14 participant_name() . . . 1914

8.238.2.15 domain_id() . . . 1914
8.238.2.16 transport_info() . . . 1915
8.238.2.17 partial_configuration() . . . 1915
8.238.2.18 reachability_lease_duration() . . . 1915
8.238.2.19 vendor_builtin_endpoints() . . . 1915
8.239 rti::test::ParticipantFactoryGuard Class Reference . . . 1916
8.239.1 Detailed Description . . . 1916
8.240 rti::topic::trust::ParticipantTrustAlgorithmInfo Class Reference . . . 1916
8.240.1 Detailed Description . . . 1916
8.240.2 Constructor & Destructor Documentation . . . 1916
8.240.2.1 ParticipantTrustAlgorithmInfo() . . . 1917
8.240.3 Member Function Documentation . . . 1917
8.240.3.1 signature() . . . 1917
8.240.3.2 key_establishment() . . . 1917
8.240.3.3 interceptor() . . . 1917
8.241 rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo Class Reference . . . 1917
8.241.1 Detailed Description . . . 1918
8.241.2 Constructor & Destructor Documentation . . . 1918
8.241.2.1 ParticipantTrustInterceptorAlgorithmInfo() . . . 1918
8.241.3 Member Function Documentation . . . 1918
8.241.3.1 supported_mask() . . . 1918
8.241.3.2 builtin_endpoints_required_mask() . . . 1919
8.241.3.3 builtin_kx_endpoints_required_mask() . . . 1919
8.241.3.4 user_endpoints_default_required_mask() . . . 1919
8.242 rti::topic::trust::ParticipantTrustKeyEstablishmentAlgorithmInfo Class Reference . . . 1919
8.242.1 Detailed Description . . . 1920
8.242.2 Constructor & Destructor Documentation . . . 1920
8.242.2.1 ParticipantTrustKeyEstablishmentAlgorithmInfo() . . . 1920
8.242.3 Member Function Documentation . . . 1920
8.242.3.1 shared_secret() . . . 1920
8.243 rti::topic::trust::ParticipantTrustProtectionInfo Class Reference . . . 1920
8.243.1 Detailed Description . . . 1921
8.243.2 Constructor & Destructor Documentation . . . 1921
8.243.2.1 ParticipantTrustProtectionInfo() . . . 1921
8.243.3 Member Function Documentation . . . 1921
8.243.3.1 bitmask() . . . 1921
8.243.3.2 plugin_bitmask() . . . 1922
8.244 rti::topic::trust::ParticipantTrustSignatureAlgorithmInfo Class Reference . . . 1922
8.244.1 Detailed Description . . . 1922

8.244.2 Constructor & Destructor Documentation . . . 1922
8.244.2.1 ParticipantTrustSignatureAlgorithmInfo() . . . 1922
8.244.3 Member Function Documentation . . . 1923
8.244.3.1 trust_chain() . . . 1923
8.244.3.2 message_auth() . . . 1923
8.245 dds::core::policy::Partition Class Reference . . . 1923
8.245.1 Detailed Description . . . 1924
8.245.2 Usage . . . 1924
8.245.3 Member Function Documentation . . . 1925
8.245.3.1 TPartition() [1/3] . . . 1925
8.245.3.2 TPartition() [2/3] . . . 1926
8.245.3.3 TPartition() [3/3] . . . 1926
8.245.3.4 name() [1/2] . . . 1926
8.245.3.5 name() [2/2] . . . 1927
8.246 rti::core::PersistentStorageSettings Class Reference . . . 1927
8.246.1 Detailed Description . . . 1928
8.246.2 Constructor & Destructor Documentation . . . 1929
8.246.2.1 PersistentStorageSettings() . . . 1929
8.246.3 Member Function Documentation . . . 1929
8.246.3.1 enable() [1/2] . . . 1929
8.246.3.2 enable() [2/2] . . . 1929
8.246.3.3 file_name() [1/3] . . . 1929
8.246.3.4 file_name() [2/3] . . . 1930
8.246.3.5 file_name() [3/3] . . . 1931
8.246.3.6 trace_file_name() [1/3] . . . 1931
8.246.3.7 trace_file_name() [2/3] . . . 1931
8.246.3.8 trace_file_name() [3/3] . . . 1932
8.246.3.9 journal_kind() [1/2] . . . 1932
8.246.3.10 journal_kind() [2/2] . . . 1933
8.246.3.11 synchronization_kind() [1/2] . . . 1933
8.246.3.12 synchronization_kind() [2/2] . . . 1933
8.246.3.13 vacuum() [1/2] . . . 1933
8.246.3.14 vacuum() [2/2] . . . 1934
8.246.3.15 restore() [1/2] . . . 1934
8.246.3.16 restore() [2/2] . . . 1934
8.246.3.17 writer_instance_cache_allocation() [1/2] . . . 1935
8.246.3.18 writer_instance_cache_allocation() [2/2] . . . 1935
8.246.3.19 writer_sample_cache_allocation() [1/2] . . . 1936
8.246.3.20 writer_sample_cache_allocation() [2/2] . . . 1936

8.246.3.21 writer_memory_state() [1/2] . . . 1937
8.246.3.22 writer_memory_state() [2/2] . . . 1937
8.246.3.23 reader_checkpoint_frequency() [1/2] . . . 1938
8.246.3.24 reader_checkpoint_frequency() [2/2] . . . 1938
8.247 rti::core::pointer< T > Class Template Reference . . . 1938
8.247.1 Detailed Description . . . 1939
8.247.2 Constructor & Destructor Documentation . . . 1939
8.247.2.1 pointer() [1/2] . . . 1939
8.247.2.2 pointer() [2/2] . . . 1939
8.247.3 Member Function Documentation . . . 1939
8.247.3.1 set() . . . 1939
8.247.3.2 get() . . . 1940
8.247.3.3 operator=() . . . 1940
8.247.3.4 operator==() . . . 1940
8.247.3.5 operator"!=() . . . 1940
8.248 dds::core::policy::policy_id< Policy > Class Template Reference . . . 1940
8.248.1 Detailed Description . . . 1940
8.249 dds::core::policy::policy_name< Policy > Class Template Reference . . . 1941
8.249.1 Detailed Description . . . 1941
8.250 dds::core::PreconditionNotMetError Class Reference . . . 1941
8.250.1 Detailed Description . . . 1942
8.250.2 Member Function Documentation . . . 1942
8.250.2.1 what() . . . 1942
8.251 dds::core::policy::Presentation Class Reference . . . 1942
8.251.1 Detailed Description . . . 1943
8.251.2 Usage . . . 1944
8.251.3 Compatibility . . . 1946
8.251.4 Constructor & Destructor Documentation . . . 1946
8.251.4.1 Presentation() [1/2] . . . 1946
8.251.4.2 Presentation() [2/2] . . . 1947
8.251.5 Member Function Documentation . . . 1947
8.251.5.1 access_scope() [1/2] . . . 1947
8.251.5.2 access_scope() [2/2] . . . 1947
8.251.5.3 coherent_access() [1/2] . . . 1948
8.251.5.4 coherent_access() [2/2] . . . 1948
8.251.5.5 ordered_access() [1/2] . . . 1948
8.251.5.6 ordered_access() [2/2] . . . 1949
8.251.5.7 GroupAccessScope() . . . 1949
8.251.5.8 InstanceAccessScope() . . . 1949

8.251.5.9 TopicAccessScope() . . . 1949
8.251.5.10 drop_incomplete_coherent_set() [1/2] . . . 1950
8.251.5.11 drop_incomplete_coherent_set() [2/2] . . . 1950
8.252 dds::core::policy::PresentationAccessScopeKind_def Struct Reference . . . 1950
8.252.1 Detailed Description . . . 1951
8.252.2 Member Enumeration Documentation . . . 1951
8.252.2.1 type . . . 1951
8.253 rti::flat::PrimitiveArrayOffset< T, N > Class Template Reference . . . 1951
8.253.1 Detailed Description . . . 1952
8.253.2 Member Function Documentation . . . 1952
8.253.2.1 element_count() . . . 1952
8.254 rti::flat::PrimitiveConstOffset< T > Struct Template Reference . . . 1952
8.254.1 Detailed Description . . . 1953
8.254.2 Member Function Documentation . . . 1953
8.254.2.1 get() . . . 1953
8.255 rti::flat::PrimitiveOffset< T > Struct Template Reference . . . 1953
8.255.1 Detailed Description . . . 1954
8.255.2 Member Function Documentation . . . 1954
8.255.2.1 set() . . . 1954
8.256 rti::flat::PrimitiveSequenceBuilder< T > Class Template Reference . . . 1955
8.256.1 Detailed Description . . . 1955
8.256.2 Member Function Documentation . . . 1955
8.256.2.1 add_next() . . . 1956
8.256.2.2 add_n() [1/3] . . . 1956
8.256.2.3 add_n() [2/3] . . . 1956
8.256.2.4 add_n() [3/3] . . . 1957
8.256.2.5 finish() . . . 1957
8.257 rti::flat::PrimitiveSequenceOffset< T > Class Template Reference . . . 1958
8.257.1 Detailed Description . . . 1958
8.257.2 Member Function Documentation . . . 1958
8.257.2.1 element_count() . . . 1958
8.258 dds::core::xtypes::PrimitiveType Class Reference . . . 1959
8.258.1 Detailed Description . . . 1959
8.258.2 Friends And Related Function Documentation . . . 1960
8.258.2.1 primitive_type() . . . 1960
8.259 rti::config::PrintFormat_def Struct Reference . . . 1960
8.259.1 Detailed Description . . . 1960
8.259.2 Member Enumeration Documentation . . . 1960
8.259.2.1 type . . . 1960

8.260 rti::topic::PrintFormatKind_def Struct Reference . . . 1961
8.260.1 Detailed Description . . . 1961
8.260.2 Member Enumeration Documentation . . . 1961
8.260.2.1 type . . . 1961
8.261 rti::topic::PrintFormatProperty Class Reference . . . 1962
8.261.1 Detailed Description . . . 1963
8.261.2 Constructor & Destructor Documentation . . . 1963
8.261.2.1 PrintFormatProperty() . . . 1963
8.261.3 Member Function Documentation . . . 1963
8.261.3.1 kind() [1/2] . . . 1963
8.261.3.2 kind() [2/2] . . . 1963
8.261.3.3 pretty_print() [1/2] . . . 1964
8.261.3.4 pretty_print() [2/2] . . . 1964
8.261.3.5 enum_as_int() [1/2] . . . 1964
8.261.3.6 enum_as_int() [2/2] . . . 1965
8.261.3.7 include_root_elements() [1/2] . . . 1965
8.261.3.8 include_root_elements() [2/2] . . . 1966
8.261.3.9 Default() . . . 1966
8.261.3.10 Xml() . . . 1966
8.261.3.11 Json() . . . 1967
8.262 rti::core::ProductVersion Class Reference . . . 1967
8.262.1 Detailed Description . . . 1967
8.262.2 Constructor & Destructor Documentation . . . 1967
8.262.2.1 ProductVersion() . . . 1968
8.262.3 Member Function Documentation . . . 1968
8.262.3.1 to_string() . . . 1968
8.262.3.2 major_version() . . . 1968
8.262.3.3 minor_version() . . . 1968
8.262.3.4 release_version() . . . 1968
8.262.3.5 revision_version() . . . 1969
8.262.3.6 current() . . . 1969
8.262.3.7 unknown() . . . 1969
8.263 rti::core::policy::Property Class Reference . . . 1969
8.263.1 Detailed Description . . . 1970
8.263.2 Usage . . . 1971
8.263.2.1 Reasons for Using the PropertyQosPolicy . . . 1971
8.263.3 Member Typedef Documentation . . . 1972
8.263.3.1 Entry . . . 1972
8.263.4 Constructor & Destructor Documentation . . . 1972

8.263.4.1 Property() [1/3] . . . 1972
8.263.4.2 Property() [2/3] . . . 1972
8.263.4.3 Property() [3/3] . . . 1973
8.263.5 Member Function Documentation . . . 1973
8.263.5.1 exists() . . . 1973
8.263.5.2 get() . . . 1973
8.263.5.3 try_get() . . . 1973
8.263.5.4 set() [1/3] . . . 1973
8.263.5.5 set() [2/3] . . . 1974
8.263.5.6 remove() . . . 1974
8.263.5.7 size() . . . 1975
8.263.5.8 get_all() . . . 1975
8.263.5.9 set() [3/3] . . . 1975
8.263.5.10 propagate() . . . 1975
8.264 rti::core::ProtocolVersion Class Reference . . . 1975
8.264.1 Detailed Description . . . 1976
8.264.2 Constructor & Destructor Documentation . . . 1976
8.264.2.1 ProtocolVersion() [1/2] . . . 1976
8.264.2.2 ProtocolVersion() [2/2] . . . 1976
8.264.3 Member Function Documentation . . . 1976
8.264.3.1 major_version() . . . 1977
8.264.3.2 minor_version() . . . 1977
8.264.3.3 current() . . . 1977
8.265 dds::topic::PublicationBuiltinTopicData Class Reference . . . 1977
8.265.1 Detailed Description . . . 1979
8.265.2 Member Function Documentation . . . 1979
8.265.2.1 TPublicationBuiltinTopicData() . . . 1979
8.265.2.2 key() . . . 1980
8.265.2.3 participant_key() . . . 1980
8.265.2.4 topic_name() . . . 1980
8.265.2.5 type_name() . . . 1980
8.265.2.6 durability() . . . 1981
8.265.2.7 durability_service() . . . 1981
8.265.2.8 deadline() . . . 1981
8.265.2.9 latency_budget() . . . 1981
8.265.2.10 liveliness() . . . 1982
8.265.2.11 reliability() . . . 1982
8.265.2.12 lifespan() . . . 1982
8.265.2.13 user_data() . . . 1982

8.265.2.14 ownership() . . . 1983
8.265.2.15 ownership_strength() . . . 1983
8.265.2.16 destination_order() . . . 1983
8.265.2.17 presentation() . . . 1983
8.265.2.18 partition() . . . 1984
8.265.2.19 topic_data() . . . 1984
8.265.2.20 group_data() . . . 1984
8.265.2.21 representation() [1/2] . . . 1985
8.265.2.22 data_tag() . . . 1985
8.265.2.23 representation() [2/2] . . . 1985
8.265.2.24 type() . . . 1986
8.265.2.25 get_type_no_copy() . . . 1986
8.265.2.26 publisher_key() . . . 1987
8.265.2.27 property() . . . 1987
8.265.2.28 unicast_locators() . . . 1987
8.265.2.29 virtual_guid() . . . 1987
8.265.2.30 rtps_protocol_version() . . . 1988
8.265.2.31 rtps_vendor_id() . . . 1988
8.265.2.32 product_version() . . . 1988
8.265.2.33 locator_filter() . . . 1988
8.265.2.34 disable_positive_acks() . . . 1989
8.265.2.35 publication_name() . . . 1989
8.265.2.36 trust_protection_info() . . . 1989
8.265.2.37 trust_algorithm_info() . . . 1990
8.265.2.38 related_datareader_key() . . . 1990
8.265.2.39 service() . . . 1990
8.266 dds::core::status::PublicationMatchedStatus Class Reference . . . 1990
8.266.1 Detailed Description . . . 1991
8.266.2 Member Function Documentation . . . 1991
8.266.2.1 total_count() . . . 1991
8.266.2.2 total_count_change() . . . 1991
8.266.2.3 current_count() . . . 1992
8.266.2.4 current_count_change() . . . 1992
8.266.2.5 last_subscription_handle() . . . 1992
8.266.2.6 current_count_peak() . . . 1992
8.267 dds::pub::Publisher Class Reference . . . 1993
8.267.1 Detailed Description . . . 1994
8.267.2 Constructor & Destructor Documentation . . . 1995
8.267.2.1 Publisher() [1/3] . . . 1995

8.267.2.2 Publisher() [2/3] . . . 1996
8.267.2.3 Publisher() [3/3] . . . 1996
8.267.3 Member Function Documentation . . . 1997
8.267.3.1 qos() [1/2] . . . 1997
8.267.3.2 qos() [2/2] . . . 1997
8.267.3.3 operator<<() . . . 1998
8.267.3.4 operator>>() . . . 1998
8.267.3.5 default_datawriter_qos() [1/2] . . . 1998
8.267.3.6 default_datawriter_qos() [2/2] . . . 1999
8.267.3.7 listener() [1/2] . . . 1999
8.267.3.8 listener() [2/2] . . . 1999
8.267.3.9 set_listener() [1/2] . . . 2000
8.267.3.10 set_listener() [2/2] . . . 2000
8.267.3.11 get_listener() . . . 2001
8.267.3.12 wait_for_acknowledgments() . . . 2001
8.267.3.13 participant() . . . 2001
8.267.3.14 wait_for_asynchronous_publishing() . . . 2002
8.267.3.15 close_contained_entities() . . . 2002
8.267.4 Friends And Related Function Documentation . . . 2003
8.267.4.1 find_publishers() [1/2] . . . 2003
8.267.4.2 find_publishers() [2/2] . . . 2004
8.267.4.3 find_publisher() . . . 2004
8.267.4.4 implicit_publisher() . . . 2005
8.268 dds::pub::PublisherListener Class Reference . . . 2006
8.268.1 Detailed Description . . . 2006
8.269 dds::pub::qos::PublisherQos Class Reference . . . 2007
8.269.1 Detailed Description . . . 2007
8.269.2 PublisherQos Policies . . . 2008
8.269.3 Member Function Documentation . . . 2008
8.269.3.1 policy() [1/3] . . . 2008
8.269.3.2 policy() [2/3] . . . 2009
8.269.3.3 policy() [3/3] . . . 2009
8.269.3.4 operator<<() . . . 2010
8.269.3.5 operator>>() . . . 2010
8.269.4 Friends And Related Function Documentation . . . 2010
8.269.4.1 swap() . . . 2010
8.269.4.2 to_string() [1/3] . . . 2011
8.269.4.3 to_string() [2/3] . . . 2012
8.269.4.4 to_string() [3/3] . . . 2012

8.269.4.5 operator<<() . . . 2013
8.270 rti::core::policy::PublishMode Class Reference . . . 2013
8.270.1 Detailed Description . . . 2014
8.270.2 Usage . . . 2014
8.270.3 Constructor & Destructor Documentation . . . 2015
8.270.3.1 PublishMode() . . . 2015
8.270.4 Member Function Documentation . . . 2015
8.270.4.1 Synchronous() . . . 2015
8.270.4.2 Asynchronous() [1/3] . . . 2015
8.270.4.3 Asynchronous() [2/3] . . . 2016
8.270.4.4 Asynchronous() [3/3] . . . 2016
8.270.4.5 kind() [1/2] . . . 2016
8.270.4.6 kind() [2/2] . . . 2016
8.270.4.7 flow_controller_name() [1/2] . . . 2016
8.270.4.8 flow_controller_name() [2/2] . . . 2017
8.270.4.9 priority() [1/2] . . . 2017
8.270.4.10 priority() [2/2] . . . 2018
8.271 rti::core::policy::PublishModeKind_def Struct Reference . . . 2018
8.271.1 Detailed Description . . . 2018
8.271.2 Member Enumeration Documentation . . . 2019
8.271.2.1 type . . . 2019
8.272 rti::core::qos_print_all_t Struct Reference . . . 2019
8.272.1 Detailed Description . . . 2020
8.273 dds::core::policy::QosPolicyCount Class Reference . . . 2020
8.273.1 Detailed Description . . . 2020
8.273.2 Constructor & Destructor Documentation . . . 2020
8.273.2.1 QosPolicyCount() . . . 2021
8.273.3 Member Function Documentation . . . 2021
8.273.3.1 policy_id() . . . 2021
8.273.3.2 count() . . . 2021
8.274 rti::core::QosPrintFormat Class Reference . . . 2021
8.274.1 Detailed Description . . . 2022
8.274.2 Constructor & Destructor Documentation . . . 2022
8.274.2.1 QosPrintFormat() [1/2] . . . 2022
8.274.2.2 QosPrintFormat() [2/2] . . . 2022
8.274.3 Member Function Documentation . . . 2022
8.274.3.1 indent() [1/2] . . . 2022
8.274.3.2 indent() [2/2] . . . 2023
8.274.3.3 print_private() [1/2] . . . 2023

8.274.3.4 print_private() [2/2] . . . 2024
8.274.3.5 is_standalone() [1/2] . . . 2024
8.274.3.6 is_standalone() [2/2] . . . 2024
8.275 dds::core::QosProvider Class Reference . . . 2025
8.275.1 Detailed Description . . . 2028
8.275.2 Selecting a QoS profile . . . 2029
8.275.3 The Default QosProvider . . . 2029
8.275.4 Built-in QoS Profiles . . . 2029
8.275.5 Specifying a Topic Filter . . . 2030
8.275.6 Constructor & Destructor Documentation . . . 2030
8.275.6.1 QosProvider() [1/2] . . . 2030
8.275.6.2 QosProvider() [2/2] . . . 2030
8.275.7 Member Function Documentation . . . 2031
8.275.7.1 participant_qos() [1/2] . . . 2031
8.275.7.2 participant_qos() [2/2] . . . 2031
8.275.7.3 topic_qos() [1/2] . . . 2032
8.275.7.4 topic_qos() [2/2] . . . 2032
8.275.7.5 subscriber_qos() [1/2] . . . 2032
8.275.7.6 subscriber_qos() [2/2] . . . 2033
8.275.7.7 datareader_qos() [1/2] . . . 2033
8.275.7.8 datareader_qos() [2/2] . . . 2034
8.275.7.9 publisher_qos() [1/2] . . . 2034
8.275.7.10 publisher_qos() [2/2] . . . 2034
8.275.7.11 datawriter_qos() [1/2] . . . 2035
8.275.7.12 datawriter_qos() [2/2] . . . 2035
8.275.7.13 Default() . . . 2036
8.275.7.14 reset_default() . . . 2036
8.275.7.15 default_provider_params() [1/2] . . . 2037
8.275.7.16 default_provider_params() [2/2] . . . 2037
8.275.7.17 participant_factory_qos() [1/2] . . . 2037
8.275.7.18 participant_factory_qos() [2/2] . . . 2038
8.275.7.19 topic_qos_w_topic_name() [1/2] . . . 2038
8.275.7.20 topic_qos_w_topic_name() [2/2] . . . 2038
8.275.7.21 datareader_qos_w_topic_name() [1/2] . . . 2039
8.275.7.22 datareader_qos_w_topic_name() [2/2] . . . 2039
8.275.7.23 datawriter_qos_w_topic_name() [1/2] . . . 2040
8.275.7.24 datawriter_qos_w_topic_name() [2/2] . . . 2040
8.275.7.25 default_library() [1/2] . . . 2041
8.275.7.26 default_profile() [1/2] . . . 2041

8.275.7.27 default_library() [2/2] . . . 2042
8.275.7.28 default_profile() [2/2] . . . 2042
8.275.7.29 default_profile_library() . . . 2043
8.275.7.30 qos_profile_libraries() . . . 2043
8.275.7.31 qos_profiles() . . . 2043
8.275.7.32 profiles_loaded() . . . 2044
8.275.7.33 type() . . . 2044
8.275.7.34 provider_params() [1/2] . . . 2044
8.275.7.35 provider_params() [2/2] . . . 2045
8.275.7.36 load_profiles() . . . 2045
8.275.7.37 reload_profiles() . . . 2045
8.275.7.38 unload_profiles() . . . 2046
8.275.7.39 create_participant_from_config() . . . 2046
8.275.8 Friends And Related Function Documentation . . . 2047
8.275.8.1 USE_DDS_DEFAULT_QOS_PROFILE . . . 2047
8.275.8.2 create_qos_provider_ex() . . . 2048
8.275.8.3 default_qos_provider_params() [1/2] . . . 2048
8.275.8.4 default_qos_provider_params() [2/2] . . . 2048
8.276 rti::core::QosProviderParams Class Reference . . . 2049
8.276.1 Detailed Description . . . 2049
8.276.2 Constructor & Destructor Documentation . . . 2050
8.276.2.1 QosProviderParams() . . . 2050
8.276.3 Member Function Documentation . . . 2050
8.276.3.1 string_profile() [1/2] . . . 2050
8.276.3.2 string_profile() [2/2] . . . 2050
8.276.3.3 url_profile() [1/2] . . . 2050
8.276.3.4 url_profile() [2/2] . . . 2051
8.276.3.5 ignore_user_profile() [1/2] . . . 2051
8.276.3.6 ignore_user_profile() [2/2] . . . 2051
8.276.3.7 ignore_environment_profile() [1/2] . . . 2052
8.276.3.8 ignore_environment_profile() [2/2] . . . 2052
8.276.3.9 ignore_resource_profile() [1/2] . . . 2053
8.276.3.10 ignore_resource_profile() [2/2] . . . 2053
8.277 dds::sub::Query Class Reference . . . 2053
8.277.1 Detailed Description . . . 2054
8.277.2 Constructor & Destructor Documentation . . . 2055
8.277.2.1 Query() [1/3] . . . 2055
8.277.2.2 Query() [2/3] . . . 2055
8.277.2.3 Query() [3/3] . . . 2056

8.277.3 Member Function Documentation . . . 2056
8.277.3.1 expression() . . . 2056
8.277.3.2 begin() [1/2] . . . 2056
8.277.3.3 end() [1/2] . . . 2057
8.277.3.4 begin() [2/2] . . . 2057
8.277.3.5 end() [2/2] . . . 2057
8.277.3.6 parameters() [1/2] . . . 2057
8.277.3.7 add_parameter() . . . 2058
8.277.3.8 parameters_length() . . . 2058
8.277.3.9 parameters() [2/2] . . . 2058
8.277.3.10 name() [1/2] . . . 2058
8.277.3.11 name() [2/2] . . . 2059
8.277.3.12 data_reader() . . . 2059
8.278 dds::sub::cond::QueryCondition Class Reference . . . 2059
8.278.1 Detailed Description . . . 2060
8.278.2 Member Typedef Documentation . . . 2060
8.278.2.1 iterator . . . 2061
8.278.2.2 const_iterator . . . 2061
8.278.3 Constructor & Destructor Documentation . . . 2061
8.278.3.1 QueryCondition() [1/2] . . . 2061
8.278.3.2 QueryCondition() [2/2] . . . 2061
8.278.4 Member Function Documentation . . . 2062
8.278.4.1 expression() . . . 2062
8.278.4.2 parameters() [1/3] . . . 2062
8.278.4.3 parameters() [2/3] . . . 2062
8.278.4.4 parameters_length() . . . 2063
8.278.4.5 parameters() [3/3] . . . 2063
8.279 rti::queuing::QueueConsumer< T > Class Template Reference . . . 2063
8.279.1 Detailed Description . . . 2065
8.279.2 Constructor & Destructor Documentation . . . 2066
8.279.2.1 QueueConsumer() [1/2] . . . 2066
8.279.2.2 QueueConsumer() [2/2] . . . 2067
8.279.3 Member Function Documentation . . . 2067
8.279.3.1 listener() [1/2] . . . 2068
8.279.3.2 listener() [2/2] . . . 2068
8.279.3.3 get_listener() . . . 2068
8.279.3.4 set_listener() . . . 2068
8.279.3.5 enable() . . . 2069
8.279.3.6 acknowledge_sample() . . . 2069

8.279.3.7 acknowledge_all() . . . 2070
8.279.3.8 reader() . . . 2070
8.279.3.9 guid() . . . 2071
8.279.3.10 receive_samples() [1/2] . . . 2071
8.279.3.11 receive_samples() [2/2] . . . 2071
8.279.3.12 take_samples() [1/2] . . . 2072
8.279.3.13 take_samples() [2/2] . . . 2072
8.279.3.14 read_samples() [1/2] . . . 2073
8.279.3.15 read_samples() [2/2] . . . 2073
8.279.3.16 wait_for_samples() [1/2] . . . 2073
8.279.3.17 wait_for_samples() [2/2] . . . 2074
8.279.3.18 send_availability() . . . 2074
8.279.3.19 has_matching_reader_queue() . . . 2075
8.280 rti::queuing::QueueConsumerListener< T > Class Template Reference . . . 2075
8.280.1 Detailed Description . . . 2076
8.280.2 Member Function Documentation . . . 2076
8.280.2.1 on_sample_available() . . . 2077
8.280.2.2 on_shared_reader_queue_matched() . . . 2077
8.281 rti::queuing::QueueConsumerParams Class Reference . . . 2077
8.281.1 Detailed Description . . . 2078
8.281.2 Constructor & Destructor Documentation . . . 2078
8.281.2.1 QueueConsumerParams() . . . 2078
8.281.3 Member Function Documentation . . . 2078
8.281.3.1 enable_availability() . . . 2078
8.282 rti::queuing::QueueEntityParams< ActualEntity > Class Template Reference . . . 2079
8.282.1 Detailed Description . . . 2079
8.282.2 Member Function Documentation . . . 2079
8.282.2.1 qos_profile() . . . 2079
8.282.2.2 shared_subscriber_name() . . . 2080
8.282.2.3 entity_name() . . . 2080
8.283 rti::queuing::QueueProducer< T > Class Template Reference . . . 2080
8.283.1 Detailed Description . . . 2081
8.283.2 Constructor & Destructor Documentation . . . 2082
8.283.2.1 QueueProducer() [1/2] . . . 2082
8.283.2.2 QueueProducer() [2/2] . . . 2083
8.283.3 Member Function Documentation . . . 2083
8.283.3.1 listener() [1/2] . . . 2084
8.283.3.2 listener() [2/2] . . . 2084
8.283.3.3 get_listener() . . . 2084

8.283.3.4 set_listener() . . . 2084
8.283.3.5 enable() . . . 2085
8.283.3.6 send_sample() [1/2] . . . 2085
8.283.3.7 send_sample() [2/2] . . . 2086
8.283.3.8 wait_for_acknowledgments() [1/2] . . . 2086
8.283.3.9 wait_for_acknowledgments() [2/2] . . . 2087
8.283.3.10 writer() . . . 2088
8.283.3.11 guid() . . . 2089
8.283.3.12 has_matching_reader_queue() . . . 2089
8.284 rti::queuing::QueueProducerListener< T > Class Template Reference . . . 2090
8.284.1 Detailed Description . . . 2091
8.284.2 Member Function Documentation . . . 2091
8.284.2.1 on_sample_acknowledged() . . . 2091
8.284.2.2 on_shared_reader_queue_matched() . . . 2092
8.285 rti::queuing::QueueProducerParams Class Reference . . . 2092
8.285.1 Detailed Description . . . 2092
8.285.2 Constructor & Destructor Documentation . . . 2093
8.285.2.1 QueueProducerParams() . . . 2093
8.285.3 Member Function Documentation . . . 2093
8.285.3.1 enable_sample_replication() . . . 2093
8.285.3.2 enable_wait_for_ack() . . . 2093
8.286 rti::queuing::QueueReplier< TReq, TRep > Class Template Reference . . . 2094
8.286.1 Detailed Description . . . 2095
8.286.2 Constructor & Destructor Documentation . . . 2096
8.286.2.1 QueueReplier() [1/2] . . . 2096
8.286.2.2 QueueReplier() [2/2] . . . 2096
8.286.3 Member Function Documentation . . . 2097
8.286.3.1 listener() [1/2] . . . 2097
8.286.3.2 listener() [2/2] . . . 2098
8.286.3.3 get_listener() . . . 2098
8.286.3.4 set_listener() . . . 2098
8.286.3.5 enable() . . . 2098
8.286.3.6 send_reply() [1/2] . . . 2099
8.286.3.7 send_reply() [2/2] . . . 2099
8.286.3.8 acknowledge_request() . . . 2100
8.286.3.9 acknowledge_all() . . . 2101
8.286.3.10 wait_for_acknowledgments() [1/2] . . . 2101
8.286.3.11 wait_for_acknowledgments() [2/2] . . . 2101
8.286.3.12 consumer() . . . 2102

8.286.3.13 producer() . . . 2102
8.286.3.14 send_availability() . . . 2102
8.286.3.15 has_matching_request_reader_queue() . . . 2102
8.286.3.16 has_matching_reply_reader_queue() . . . 2103
8.286.3.17 wait_for_requests() [1/2] . . . 2103
8.286.3.18 wait_for_requests() [2/2] . . . 2103
8.286.3.19 receive_requests() [1/2] . . . 2104
8.286.3.20 receive_requests() [2/2] . . . 2104
8.286.3.21 get_write_params_for_related_request() . . . 2104
8.286.3.22 take_requests() [1/2] . . . 2105
8.286.3.23 take_requests() [2/2] . . . 2105
8.286.3.24 read_requests() [1/2] . . . 2106
8.286.3.25 read_requests() [2/2] . . . 2106
8.286.3.26 writer() . . . 2106
8.286.3.27 reader() . . . 2106
8.286.3.28 guid() . . . 2107
8.287 rti::queuing::QueueReplierListener< TReq, TRep > Class Template Reference . . . 2107
8.287.1 Detailed Description . . . 2108
8.287.2 Member Function Documentation . . . 2108
8.287.2.1 on_reply_acknowledged() . . . 2108
8.287.2.2 on_request_available() . . . 2108
8.287.2.3 on_request_shared_reader_queue_matched() . . . 2109
8.287.2.4 on_reply_shared_reader_queue_matched() . . . 2109
8.288 rti::queuing::QueueReplierParams Class Reference . . . 2109
8.288.1 Detailed Description . . . 2110
8.288.2 Constructor & Destructor Documentation . . . 2110
8.288.2.1 QueueReplierParams() . . . 2110
8.288.3 Member Function Documentation . . . 2110
8.288.3.1 enable_availability() . . . 2110
8.288.3.2 enable_sample_replication() . . . 2111
8.288.3.3 enable_wait_for_ack() . . . 2111
8.289 rti::queuing::QueueRequester< TReq, TRep > Class Template Reference . . . 2111
8.289.1 Detailed Description . . . 2113
8.289.2 Constructor & Destructor Documentation . . . 2114
8.289.2.1 QueueRequester() [1/2] . . . 2114
8.289.2.2 QueueRequester() [2/2] . . . 2114
8.289.3 Member Function Documentation . . . 2115
8.289.3.1 listener() [1/2] . . . 2115
8.289.3.2 listener() [2/2] . . . 2115

8.289.3.3 get_listener() . . . 2116
8.289.3.4 set_listener() . . . 2116
8.289.3.5 enable() . . . 2116
8.289.3.6 receive_replies() [1/2] . . . 2117
8.289.3.7 receive_replies() [2/2] . . . 2117
8.289.3.8 take_replies() [1/4] . . . 2117
8.289.3.9 take_replies() [2/4] . . . 2118
8.289.3.10 read_replies() [1/3] . . . 2118
8.289.3.11 read_replies() [2/3] . . . 2118
8.289.3.12 take_replies() [3/4] . . . 2118
8.289.3.13 take_replies() [4/4] . . . 2119
8.289.3.14 read_replies() [3/3] . . . 2119
8.289.3.15 send_request() [1/2] . . . 2120
8.289.3.16 send_request() [2/2] . . . 2120
8.289.3.17 acknowledge_reply() . . . 2121
8.289.3.18 acknowledge_all() . . . 2121
8.289.3.19 wait_for_acknowledgments() [1/2] . . . 2122
8.289.3.20 wait_for_acknowledgments() [2/2] . . . 2122
8.289.3.21 consumer() . . . 2122
8.289.3.22 producer() . . . 2123
8.289.3.23 wait_for_replies() [1/3] . . . 2123
8.289.3.24 wait_for_replies() [2/3] . . . 2123
8.289.3.25 wait_for_replies() [3/3] . . . 2123
8.289.3.26 send_availability() . . . 2124
8.289.3.27 has_matching_request_reader_queue() . . . 2125
8.289.3.28 has_matching_reply_reader_queue() . . . 2125
8.289.3.29 writer() . . . 2125
8.289.3.30 reader() . . . 2125
8.289.3.31 guid() . . . 2126
8.290 rti::queuing::QueueRequesterListener< TReq, TRep > Class Template Reference . . . 2126
8.290.1 Detailed Description . . . 2127
8.290.2 Member Function Documentation . . . 2127
8.290.2.1 on_request_acknowledged() . . . 2127
8.290.2.2 on_reply_available() . . . 2127
8.290.2.3 on_request_shared_reader_queue_matched() . . . 2128
8.290.2.4 on_reply_shared_reader_queue_matched() . . . 2128
8.291 rti::queuing::QueueRequesterParams Class Reference . . . 2128
8.291.1 Detailed Description . . . 2129
8.291.2 Constructor & Destructor Documentation . . . 2129

8.291.2.1 QueueRequesterParams() . . . 2129
8.291.3 Member Function Documentation . . . 2129
8.291.3.1 enable_availability() . . . 2129
8.291.3.2 enable_sample_replication() . . . 2130
8.291.3.3 enable_wait_for_ack() . . . 2130
8.292 dds::sub::Rank Class Reference . . . 2130
8.292.1 Detailed Description . . . 2131
8.292.2 Constructor & Destructor Documentation . . . 2131
8.292.2.1 Rank() [1/2] . . . 2131
8.292.2.2 Rank() [2/2] . . . 2131
8.292.3 Member Function Documentation . . . 2132
8.292.3.1 sample() . . . 2132
8.292.3.2 generation() . . . 2132
8.292.3.3 absolute_generation() . . . 2132
8.293 dds::sub::cond::ReadCondition Class Reference . . . 2133
8.293.1 Detailed Description . . . 2134
8.293.2 Constructor & Destructor Documentation . . . 2134
8.293.2.1 ReadCondition() [1/2] . . . 2134
8.293.2.2 ReadCondition() [2/2] . . . 2135
8.293.3 Member Function Documentation . . . 2135
8.293.3.1 state_filter() . . . 2136
8.293.3.2 data_reader() . . . 2136
8.293.3.3 close() . . . 2136
8.293.3.4 closed() . . . 2136
8.293.4 Friends And Related Function Documentation . . . 2136
8.293.4.1 create_read_condition_ex() [1/2] . . . 2137
8.293.4.2 create_read_condition_ex() [2/2] . . . 2137
8.294 dds::core::policy::ReaderDataLifecycle Class Reference . . . 2137
8.294.1 Detailed Description . . . 2138
8.294.2 Constructor & Destructor Documentation . . . 2139
8.294.2.1 ReaderDataLifecycle() [1/2] . . . 2140
8.294.2.2 ReaderDataLifecycle() [2/2] . . . 2140
8.294.3 Member Function Documentation . . . 2140
8.294.3.1 autopurge_nowriter_samples_delay() [1/2] . . . 2140
8.294.3.2 autopurge_nowriter_samples_delay() [2/2] . . . 2141
8.294.3.3 autopurge_disposed_samples_delay() [1/2] . . . 2141
8.294.3.4 autopurge_disposed_samples_delay() [2/2] . . . 2141
8.294.3.5 NoAutoPurge() . . . 2141
8.294.3.6 AutoPurgeDisposedSamples() . . . 2142

8.294.3.7 AutoPurgeNoWriterSamples() . . . 2142
8.294.3.8 autopurge_disposed_instances_delay() [1/2] . . . 2142
8.294.3.9 autopurge_disposed_instances_delay() [2/2] . . . 2143
8.294.3.10 autopurge_nowriter_instances_delay() [1/2] . . . 2143
8.294.3.11 autopurge_nowriter_instances_delay() [2/2] . . . 2143
8.295 dds::sub::ReadModeDummyType Class Reference . . . 2144
8.295.1 Detailed Description . . . 2144
8.296 rti::core::policy::ReceiverPool Class Reference . . . 2144
8.296.1 Detailed Description . . . 2145
8.296.2 Usage . . . 2145
8.296.3 Constructor & Destructor Documentation . . . 2145
8.296.3.1 ReceiverPool() [1/2] . . . 2145
8.296.3.2 ReceiverPool() [2/2] . . . 2146
8.296.4 Member Function Documentation . . . 2146
8.296.4.1 thread() [1/3] . . . 2146
8.296.4.2 thread() [2/3] . . . 2147
8.296.4.3 thread() [3/3] . . . 2147
8.296.4.4 buffer_size() [1/2] . . . 2147
8.296.4.5 buffer_size() [2/2] . . . 2148
8.296.4.6 buffer_alignment() [1/2] . . . 2148
8.296.4.7 buffer_alignment() [2/2] . . . 2148
8.297 dds::core::Reference< DELEGATE > Class Template Reference . . . 2149
8.297.1 Detailed Description . . . 2149
8.298 dds::core::policy::Reliability Class Reference . . . 2149
8.298.1 Detailed Description . . . 2150
8.298.2 Usage . . . 2151
8.298.3 Compatibility . . . 2152
8.298.4 Constructor & Destructor Documentation . . . 2152
8.298.4.1 Reliability() [1/2] . . . 2152
8.298.4.2 Reliability() [2/2] . . . 2152
8.298.5 Member Function Documentation . . . 2153
8.298.5.1 kind() [1/2] . . . 2153
8.298.5.2 kind() [2/2] . . . 2153
8.298.5.3 max_blocking_time() [1/2] . . . 2153
8.298.5.4 max_blocking_time() [2/2] . . . 2154
8.298.5.5 Reliable() . . . 2154
8.298.5.6 BestEffort() . . . 2154
8.298.5.7 acknowledgment_kind() [1/2] . . . 2154
8.298.5.8 acknowledgment_kind() [2/2] . . . 2155

8.298.5.9 instance_state_consistency_kind() [1/2] . . . 2155
8.298.5.10 instance_state_consistency_kind() [2/2] . . . 2155
8.298.5.11 exact_reliability_kind_match() [1/2] . . . 2156
8.298.5.12 exact_reliability_kind_match() [2/2] . . . 2156
8.299 dds::core::policy::ReliabilityKind_def Struct Reference . . . 2156
8.299.1 Detailed Description . . . 2157
8.299.2 Member Enumeration Documentation . . . 2157
8.299.2.1 type . . . 2157
8.300 rti::core::status::ReliableReaderActivityChangedStatus Class Reference . . . 2157
8.300.1 Detailed Description . . . 2158
8.300.2 Member Function Documentation . . . 2158
8.300.2.1 active_count() . . . 2158
8.300.2.2 inactive_count() . . . 2159
8.300.2.3 last_instance_handle() . . . 2159
8.301 rti::core::status::ReliableWriterCacheChangedStatus Class Reference . . . 2159
8.301.1 Detailed Description . . . 2160
8.301.2 Member Function Documentation . . . 2160
8.301.2.1 empty_reliable_writer_cache() . . . 2160
8.301.2.2 full_reliable_writer_cache() . . . 2160
8.301.2.3 low_watermark_reliable_writer_cache() . . . 2161
8.301.2.4 high_watermark_reliable_writer_cache() . . . 2161
8.301.2.5 unacknowledged_sample_count() . . . 2161
8.301.2.6 unacknowledged_sample_count_peak() . . . 2161
8.301.2.7 replaced_unacknowledged_sample_count() . . . 2162
8.302 dds::rpc::RemoteInvalidArgumentError Class Reference . . . 2162
8.302.1 Detailed Description . . . 2162
8.302.2 Member Function Documentation . . . 2162
8.302.2.1 what() . . . 2163
8.303 dds::rpc::RemoteOutOfResourcesError Class Reference . . . 2163
8.303.1 Detailed Description . . . 2163
8.303.2 Member Function Documentation . . . 2163
8.303.2.1 what() . . . 2164
8.304 rti::core::policy::RemoteParticipantPurgeKind_def Struct Reference . . . 2164
8.304.1 Detailed Description . . . 2164
8.304.2 Member Enumeration Documentation . . . 2164
8.304.2.1 type . . . 2164
8.305 dds::rpc::RemoteUnknownExceptionError Class Reference . . . 2165
8.305.1 Detailed Description . . . 2166
8.305.2 Member Function Documentation . . . 2166

8.305.2.1 what() . . . 2166
8.306 dds::rpc::RemoteUnknownOperationError Class Reference . . . 2166
8.306.1 Detailed Description . . . 2167
8.306.2 Member Function Documentation . . . 2167
8.306.2.1 what() . . . 2167
8.307 dds::rpc::RemoteUnsupportedOperationError Class Reference . . . 2167
8.307.1 Detailed Description . . . 2168
8.307.2 Member Function Documentation . . . 2168
8.307.2.1 what() . . . 2168
8.308 rti::request::Replier< RequestType, ReplyType > Class Template Reference . . . 2168
8.308.1 Detailed Description . . . 2170
8.308.2 Constructor & Destructor Documentation . . . 2171
8.308.2.1 Replier() [1/4] . . . 2171
8.308.2.2 Replier() [2/4] . . . 2171
8.308.2.3 Replier() [3/4] . . . 2172
8.308.2.4 Replier() [4/4] . . . 2172
8.308.3 Member Function Documentation . . . 2173
8.308.3.1 send_reply() [1/3] . . . 2173
8.308.3.2 send_reply() [2/3] . . . 2173
8.308.3.3 send_reply() [3/3] . . . 2174
8.308.3.4 wait_for_requests() [1/2] . . . 2174
8.308.3.5 wait_for_requests() [2/2] . . . 2175
8.308.3.6 receive_requests() [1/2] . . . 2175
8.308.3.7 receive_requests() [2/2] . . . 2176
8.308.3.8 take_requests() . . . 2176
8.308.3.9 read_requests() . . . 2176
8.308.3.10 listener() [1/2] . . . 2177
8.308.3.11 listener() [2/2] . . . 2177
8.308.3.12 get_listener() . . . 2177
8.308.3.13 set_listener() . . . 2177
8.308.3.14 request_datareader() . . . 2178
8.308.3.15 reply_datawriter() . . . 2178
8.308.4 Friends And Related Function Documentation . . . 2178
8.308.4.1 matched_requester_count() . . . 2179
8.309 rti::request::ReplierListener< RequestType, ReplyType > Class Template Reference . . . 2179
8.309.1 Detailed Description . . . 2179
8.309.2 Member Function Documentation . . . 2180
8.309.2.1 on_request_available() . . . 2180
8.310 rti::request::ReplierParams Class Reference . . . 2180

8.310.1 Detailed Description . . . 2181
8.310.2 Constructor & Destructor Documentation . . . 2181
8.310.2.1 ReplierParams() . . . 2181
8.310.3 Member Function Documentation . . . 2181
8.310.3.1 service_name() . . . 2181
8.310.3.2 request_topic_name() . . . 2182
8.310.3.3 reply_topic_name() . . . 2182
8.310.3.4 datawriter_qos() . . . 2182
8.310.3.5 datareader_qos() . . . 2182
8.310.3.6 publisher() . . . 2183
8.310.3.7 subscriber() . . . 2183
8.310.3.8 request_type() . . . 2183
8.310.3.9 reply_type() . . . 2183
8.310.3.10 service_instance_name() . . . 2184
8.311 dds::core::status::RequestedDeadlineMissedStatus Class Reference . . . 2184
8.311.1 Detailed Description . . . 2184
8.311.2 Member Function Documentation . . . 2184
8.311.2.1 total_count() . . . 2184
8.311.2.2 total_count_change() . . . 2185
8.311.2.3 last_instance_handle() . . . 2185
8.312 dds::core::status::RequestedIncompatibleQosStatus Class Reference . . . 2185
8.312.1 Detailed Description . . . 2186
8.312.2 Member Function Documentation . . . 2186
8.312.2.1 total_count() . . . 2186
8.312.2.2 total_count_change() . . . 2186
8.312.2.3 last_policy_id() . . . 2186
8.312.2.4 policies() . . . 2187
8.313 rti::request::Requester< RequestType, ReplyType > Class Template Reference . . . 2187
8.313.1 Detailed Description . . . 2188
8.313.2 Constructor & Destructor Documentation . . . 2189
8.313.2.1 Requester() [1/2] . . . 2189
8.313.2.2 Requester() [2/2] . . . 2190
8.313.3 Member Function Documentation . . . 2190
8.313.3.1 wait_for_service() . . . 2190
8.313.3.2 send_request() [1/2] . . . 2191
8.313.3.3 send_request() [2/2] . . . 2192
8.313.3.4 receive_replies() [1/2] . . . 2192
8.313.3.5 receive_replies() [2/2] . . . 2193
8.313.3.6 take_replies() [1/2] . . . 2193

8.313.3.7 take_replies() [2/2] . . . 2194
8.313.3.8 read_replies() [1/2] . . . 2194
8.313.3.9 read_replies() [2/2] . . . 2195
8.313.3.10 wait_for_replies() [1/3] . . . 2195
8.313.3.11 wait_for_replies() [2/3] . . . 2195
8.313.3.12 wait_for_replies() [3/3] . . . 2196
8.313.3.13 request_datawriter() . . . 2197
8.313.3.14 reply_datareader() . . . 2198
8.313.4 Friends And Related Function Documentation . . . 2198
8.313.4.1 matched_replier_count() . . . 2198
8.314 rti::request::RequesterParams Class Reference . . . 2199
8.314.1 Detailed Description . . . 2199
8.314.2 Constructor & Destructor Documentation . . . 2200
8.314.2.1 RequesterParams() . . . 2200
8.314.3 Member Function Documentation . . . 2200
8.314.3.1 service_name() . . . 2200
8.314.3.2 request_topic_name() . . . 2201
8.314.3.3 reply_topic_name() . . . 2201
8.314.3.4 datawriter_qos() . . . 2201
8.314.3.5 datareader_qos() . . . 2201
8.314.3.6 publisher() . . . 2202
8.314.3.7 subscriber() . . . 2202
8.314.3.8 request_type() . . . 2202
8.314.3.9 reply_type() . . . 2202
8.314.3.10 require_matching_service_on_send_request() [1/2] . . . 2202
8.314.3.11 require_matching_service_on_send_request() [2/2] . . . 2203
8.315 dds::core::policy::ResourceLimits Class Reference . . . 2203
8.315.1 Detailed Description . . . 2204
8.315.2 Usage . . . 2204
8.315.3 Consistency . . . 2205
8.315.4 Constructor & Destructor Documentation . . . 2205
8.315.4.1 ResourceLimits() [1/2] . . . 2205
8.315.4.2 ResourceLimits() [2/2] . . . 2205
8.315.5 Member Function Documentation . . . 2205
8.315.5.1 max_samples() [1/2] . . . 2206
8.315.5.2 max_samples() [2/2] . . . 2206
8.315.5.3 max_instances() [1/2] . . . 2206
8.315.5.4 max_instances() [2/2] . . . 2207
8.315.5.5 max_samples_per_instance() [1/2] . . . 2207

8.315.5.6 max_samples_per_instance() [2/2] . . . 2207
8.315.5.7 initial_samples() [1/2] . . . 2207
8.315.5.8 initial_samples() [2/2] . . . 2208
8.315.5.9 initial_instances() [1/2] . . . 2208
8.315.5.10 initial_instances() [2/2] . . . 2208
8.315.5.11 instance_hash_buckets() [1/2] . . . 2209
8.315.5.12 instance_hash_buckets() [2/2] . . . 2209
8.316 rti::core::Result< T > Class Template Reference . . . 2209
8.316.1 Detailed Description . . . 2210
8.316.2 Constructor & Destructor Documentation . . . 2210
8.316.2.1 Result() [1/4] . . . 2211
8.316.2.2 Result() [2/4] . . . 2211
8.316.2.3 Result() [3/4] . . . 2211
8.316.2.4 Result() [4/4] . . . 2211
8.316.3 Member Function Documentation . . . 2211
8.316.3.1 get() [1/2] . . . 2212
8.316.3.2 get() [2/2] . . . 2212
8.316.3.3 get_return_code() . . . 2212
8.316.3.4 is_ok() . . . 2212
8.316.3.5 get_if_ok() [1/3] . . . 2212
8.316.3.6 get_if_ok() [2/3] . . . 2213
8.316.3.7 get_if_ok() [3/3] . . . 2213
8.316.3.8 throw_if_error() . . . 2213
8.317 rpc_example::RobotControl Class Reference . . . 2213
8.317.1 Detailed Description . . . 2214
8.317.2 Constructor & Destructor Documentation . . . 2214
8.317.2.1 ~RobotControl() . . . 2214
8.317.3 Member Function Documentation . . . 2214
8.317.3.1 walk_to() . . . 2214
8.317.3.2 get_speed() . . . 2215
8.318 rpc_example::RobotControlAsync Class Reference . . . 2215
8.318.1 Detailed Description . . . 2216
8.318.2 Constructor & Destructor Documentation . . . 2216
8.318.2.1 ~RobotControlAsync() . . . 2216
8.318.3 Member Function Documentation . . . 2216
8.318.3.1 walk_to_async() . . . 2216
8.318.3.2 get_speed_async() . . . 2217
8.319 rpc_example::RobotControlClient Class Reference . . . 2217
8.319.1 Detailed Description . . . 2218

8.320 rti::config::compliance::RpcMask Class Reference 2218
8.320.1 Detailed Description 2219
8.320.2 Member Typedef Documentation 2219
8.320.2.1 MaskType 2219
8.320.3 Constructor & Destructor Documentation 2219
8.320.3.1 RpcMask() [1/3] 2220
8.320.3.2 RpcMask() [2/3] 2220
8.320.3.3 RpcMask() [3/3] 2220
8.320.4 Member Function Documentation 2220
8.320.4.1 use_standard_related_sample_pid() 2220
8.320.4.2 vendor() 2221
8.320.4.3 default_mask() 2221
8.321 rti::core::RtpsReliableReaderProtocol Class Reference 2221
8.321.1 Detailed Description 2223
8.321.2 Constructor & Destructor Documentation 2223
8.321.2.1 RtpsReliableReaderProtocol() 2223
8.321.3 Member Function Documentation 2223
8.321.3.1 min_heartbeat_response_delay() [1/2] 2223
8.321.3.2 min_heartbeat_response_delay() [2/2] 2224
8.321.3.3 max_heartbeat_response_delay() [1/2] 2224
8.321.3.4 max_heartbeat_response_delay() [2/2] 2224
8.321.3.5 heartbeat_suppression_duration() [1/2] 2225
8.321.3.6 heartbeat_suppression_duration() [2/2] 2225
8.321.3.7 nack_period() [1/2] 2225
8.321.3.8 nack_period() [2/2] 2226
8.321.3.9 receive_window_size() [1/2] 2226
8.321.3.10 receive_window_size() [2/2] 2226
8.321.3.11 round_trip_time() [1/2] 2226
8.321.3.12 round_trip_time() [2/2] 2227
8.321.3.13 app_ack_period() [1/2] 2227
8.321.3.14 app_ack_period() [2/2] 2227
8.321.3.15 min_app_ack_response_keep_duration() [1/2] 2228
8.321.3.16 min_app_ack_response_keep_duration() [2/2] 2228
8.321.3.17 samples_per_app_ack() [1/2] 2228
8.321.3.18 samples_per_app_ack() [2/2] 2229
8.322 rti::core::policy::RtpsReliableWriterProtocol Class Reference 2229
8.322.1 Detailed Description 2232
8.322.2 Constructor & Destructor Documentation 2232
8.322.2.1 RtpsReliableWriterProtocol() 2232

8.322.3 Member Function Documentation 2232
8.322.3.1 low_watermark() [1/2] 2233
8.322.3.2 low_watermark() [2/2] 2233
8.322.3.3 high_watermark() [1/2] 2234
8.322.3.4 high_watermark() [2/2] 2234
8.322.3.5 heartbeat_period() [1/2] 2235
8.322.3.6 heartbeat_period() [2/2] 2235
8.322.3.7 fast_heartbeat_period() [1/2] 2236
8.322.3.8 fast_heartbeat_period() [2/2] 2236
8.322.3.9 late_joiner_heartbeat_period() [1/2] 2237
8.322.3.10 late_joiner_heartbeat_period() [2/2] 2237
8.322.3.11 virtual_heartbeat_period() 2238
8.322.3.12 samples_per_virtual_heartbeat() [1/2] 2238
8.322.3.13 samples_per_virtual_heartbeat() [2/2] 2239
8.322.3.14 max_heartbeat_retries() [1/2] 2239
8.322.3.15 max_heartbeat_retries() [2/2] 2239
8.322.3.16 inactivate_nonprogressing_readers() [1/2] 2240
8.322.3.17 inactivate_nonprogressing_readers() [2/2] 2240
8.322.3.18 heartbeats_per_max_samples() [1/2] 2240
8.322.3.19 heartbeats_per_max_samples() [2/2] 2242
8.322.3.20 min_nack_response_delay() [1/2] 2242
8.322.3.21 min_nack_response_delay() [2/2] 2242
8.322.3.22 max_nack_response_delay() [1/2] 2243
8.322.3.23 max_nack_response_delay() [2/2] 2243
8.322.3.24 nack_suppression_duration() [1/2] 2244
8.322.3.25 nack_suppression_duration() [2/2] 2244
8.322.3.26 max_bytes_per_nack_response() [1/2] 2245
8.322.3.27 max_bytes_per_nack_response() [2/2] 2245
8.322.3.28 disable_positive_acks_min_sample_keep_duration() [1/2] 2246
8.322.3.29 disable_positive_acks_min_sample_keep_duration() [2/2] 2246
8.322.3.30 disable_positive_acks_max_sample_keep_duration() [1/2] 2246
8.322.3.31 disable_positive_acks_max_sample_keep_duration() [2/2] 2247
8.322.3.32 disable_positive_acks_enable_adaptive_sample_keep_duration() [1/2] 2247
8.322.3.33 disable_positive_acks_enable_adaptive_sample_keep_duration() [2/2] 2248
8.322.3.34 disable_positive_acks_decrease_sample_keep_duration_factor() [1/2] 2248
8.322.3.35 disable_positive_acks_decrease_sample_keep_duration_factor() [2/2] 2248
8.322.3.36 disable_positive_acks_increase_sample_keep_duration_factor() [1/2] 2249
8.322.3.37 disable_positive_acks_increase_sample_keep_duration_factor() [2/2] 2249
8.322.3.38 min_send_window_size() [1/2] 2249

8.322.3.39 min_send_window_size() [2/2] . . . 2250
8.322.3.40 max_send_window_size() [1/2] . . . 2250
8.322.3.41 max_send_window_size() [2/2] . . . 2251
8.322.3.42 send_window_update_period() [1/2] . . . 2251
8.322.3.43 send_window_update_period() [2/2] . . . 2252
8.322.3.44 send_window_increase_factor() [1/2] . . . 2252
8.322.3.45 send_window_increase_factor() [2/2] . . . 2253
8.322.3.46 send_window_decrease_factor() [1/2] . . . 2253
8.322.3.47 send_window_decrease_factor() [2/2] . . . 2254
8.322.3.48 enable_multicast_periodic_heartbeat() [1/2] . . . 2254
8.322.3.49 enable_multicast_periodic_heartbeat() [2/2] . . . 2254
8.322.3.50 multicast_resend_threshold() [1/2] . . . 2254
8.322.3.51 multicast_resend_threshold() [2/2] . . . 2255
8.322.3.52 disable_repair_piggyback_heartbeat() [1/2] . . . 2255
8.322.3.53 disable_repair_piggyback_heartbeat() [2/2] . . . 2255
8.323 rti::core::policy::RtpsReservedPortKindMask Class Reference . . . 2255
8.323.1 Detailed Description . . . 2256
8.323.2 Member Typedef Documentation . . . 2256
8.323.2.1 MaskType . . . 2257
8.323.3 Constructor & Destructor Documentation . . . 2257
8.323.3.1 RtpsReservedPortKindMask() [1/3] . . . 2257
8.323.3.2 RtpsReservedPortKindMask() [2/3] . . . 2257
8.323.3.3 RtpsReservedPortKindMask() [3/3] . . . 2257
8.323.4 Member Function Documentation . . . 2257
8.323.4.1 builtin_unicast() . . . 2258
8.323.4.2 builtin_multicast() . . . 2258
8.323.4.3 user_unicast() . . . 2258
8.323.4.4 user_multicast() . . . 2258
8.324 rti::core::RtpsWellKnownPorts Class Reference . . . 2258
8.324.1 Detailed Description . . . 2260
8.324.2 Constructor & Destructor Documentation . . . 2261
8.324.2.1 RtpsWellKnownPorts() [1/2] . . . 2261
8.324.2.2 RtpsWellKnownPorts() [2/2] . . . 2261
8.324.3 Member Function Documentation . . . 2262
8.324.3.1 port_base() [1/2] . . . 2262
8.324.3.2 port_base() [2/2] . . . 2262
8.324.3.3 domain_id_gain() [1/2] . . . 2262
8.324.3.4 domain_id_gain() [2/2] . . . 2262
8.324.3.5 participant_id_gain() [1/2] . . . 2263

8.324.3.6 participant_id_gain() [2/2] . . . 2263
8.324.3.7 builtin_multicast_port_offset() [1/2] . . . 2264
8.324.3.8 builtin_multicast_port_offset() [2/2] . . . 2264
8.324.3.9 builtin_unicast_port_offset() [1/2] . . . 2264
8.324.3.10 builtin_unicast_port_offset() [2/2] . . . 2264
8.324.3.11 user_multicast_port_offset() [1/2] . . . 2264
8.324.3.12 user_multicast_port_offset() [2/2] . . . 2265
8.324.3.13 user_unicast_port_offset() [1/2] . . . 2265
8.324.3.14 user_unicast_port_offset() [2/2] . . . 2265
8.325 dds::core::safe_enum< def, inner > Class Template Reference . . . 2265
8.325.1 Detailed Description . . . 2266
8.325.2 Member Typedef Documentation . . . 2267
8.325.2.1 inner_enum . . . 2267
8.325.3 Constructor & Destructor Documentation . . . 2267
8.325.3.1 safe_enum() [1/2] . . . 2267
8.325.3.2 safe_enum() [2/2] . . . 2268
8.325.4 Member Function Documentation . . . 2268
8.325.4.1 underlying() . . . 2268
8.325.4.2 operator==() . . . 2268
8.325.4.3 operator"!=() . . . 2268
8.325.4.4 operator<() . . . 2269
8.325.4.5 operator<=() . . . 2269
8.325.4.6 operator>() . . . 2269
8.325.4.7 operator>=() . . . 2269
8.325.5 Friends And Related Function Documentation . . . 2269
8.325.5.1 swap . . . 2270
8.325.5.2 operator<<() . . . 2270
8.326 dds::sub::Sample< T > Class Template Reference . . . 2270
8.326.1 Detailed Description . . . 2271
8.326.2 Constructor & Destructor Documentation . . . 2271
8.326.2.1 Sample() [1/4] . . . 2271
8.326.2.2 Sample() [2/4] . . . 2271
8.326.2.3 Sample() [3/4] . . . 2272
8.326.2.4 Sample() [4/4] . . . 2272
8.326.3 Member Function Documentation . . . 2272
8.326.3.1 operator=() [1/2] . . . 2272
8.326.3.2 data() [1/2] . . . 2272
8.326.3.3 data() [2/2] . . . 2273
8.326.3.4 info() [1/2] . . . 2273

8.326.3.5 info() [2/2] . . . 2273
8.326.3.6 operator=() [2/2] . . . 2273
8.327 rti::flat::Sample< OffsetType > Class Template Reference . . . 2274
8.327.1 Detailed Description . . . 2274
8.327.2 Member Typedef Documentation . . . 2275
8.327.2.1 Offset . . . 2275
8.327.2.2 ConstOffset . . . 2275
8.327.3 Member Function Documentation . . . 2276
8.327.3.1 root() [1/2] . . . 2276
8.327.3.2 root() [2/2] . . . 2276
8.327.3.3 create_data() . . . 2277
8.327.3.4 clone() . . . 2277
8.327.3.5 delete_data() . . . 2277
8.328 rti::core::SampleFlag Class Reference . . . 2278
8.328.1 Detailed Description . . . 2279
8.328.2 Member Typedef Documentation . . . 2279
8.328.2.1 MaskType . . . 2279
8.328.3 Constructor & Destructor Documentation . . . 2279
8.328.3.1 SampleFlag() [1/3] . . . 2279
8.328.3.2 SampleFlag() [2/3] . . . 2279
8.328.3.3 SampleFlag() [3/3] . . . 2280
8.328.4 Member Function Documentation . . . 2280
8.328.4.1 redelivered() . . . 2280
8.328.4.2 intermediate_reply_sequence() . . . 2280
8.328.4.3 replicate() . . . 2280
8.328.4.4 last_shared_reader_queue() . . . 2281
8.328.4.5 intermediate_topic_query_sample() . . . 2281
8.328.4.6 incomplete_snapshot_topic_query() . . . 2281
8.328.4.7 writer_removed_batch_sample() . . . 2281
8.328.4.8 discovery_service_sample() . . . 2282
8.329 rti::core::SampleIdentity Class Reference . . . 2282
8.329.1 Detailed Description . . . 2283
8.329.2 Constructor & Destructor Documentation . . . 2283
8.329.2.1 SampleIdentity() [1/2] . . . 2283
8.329.2.2 SampleIdentity() [2/2] . . . 2283
8.329.3 Member Function Documentation . . . 2283
8.329.3.1 writer_guid() [1/2] . . . 2283
8.329.3.2 writer_guid() [2/2] . . . 2284
8.329.3.3 sequence_number() [1/2] . . . 2284

8.329.3.4 sequence_number() [2/2] . . . 2284
8.329.3.5 automatic() . . . 2284
8.329.3.6 unknown() . . . 2284
8.329.4 Friends And Related Function Documentation . . . 2284
8.329.4.1 operator<<() . . . 2285
8.330 dds::sub::SampleInfo Class Reference . . . 2285
8.330.1 Detailed Description . . . 2286
8.330.2 Interpretation of the SampleInfo . . . 2286
8.330.3 Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count 2287
8.330.4 Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank 2288
8.330.5 Interpretation of the SampleInfo counters and ranks . . . 2288
8.330.6 Member Function Documentation . . . 2289
8.330.6.1 source_timestamp() . . . 2289
8.330.6.2 state() . . . 2289
8.330.6.3 generation_count() . . . 2289
8.330.6.4 rank() . . . 2289
8.330.6.5 valid() . . . 2290
8.330.6.6 instance_handle() . . . 2290
8.330.6.7 publication_handle() . . . 2290
8.330.6.8 reception_timestamp() . . . 2291
8.330.6.9 publication_sequence_number() . . . 2291
8.330.6.10 reception_sequence_number() . . . 2291
8.330.6.11 original_publication_virtual_guid() . . . 2291
8.330.6.12 original_publication_virtual_sequence_number() . . . 2292
8.330.6.13 original_publication_virtual_sample_identity() . . . 2292
8.330.6.14 related_original_publication_virtual_guid() . . . 2292
8.330.6.15 related_original_publication_virtual_sequence_number() . . . 2293
8.330.6.16 related_original_publication_virtual_sample_identity() . . . 2293
8.330.6.17 flag() . . . 2293
8.330.6.18 source_guid() . . . 2294
8.330.6.19 related_source_guid() . . . 2294
8.330.6.20 related_subscription_guid() . . . 2294
8.330.6.21 topic_query_guid() . . . 2295
8.330.6.22 encapsulation_id() . . . 2295
8.330.6.23 coherent_set_info() . . . 2295
8.331 rti::sub::SampleIterator< T > Class Template Reference . . . 2296
8.331.1 Detailed Description . . . 2296
8.331.2 Member Function Documentation . . . 2296
8.331.2.1 operator++() . . . 2296

8.332 rti::core::status::SampleLostState Class Reference . . . 2296
8.332.1 Detailed Description . . . 2298
8.332.2 Constructor & Destructor Documentation . . . 2298
8.332.2.1 SampleLostState() [1/2] . . . 2298
8.332.2.2 SampleLostState() [2/2] . . . 2298
8.332.3 Member Function Documentation . . . 2299
8.332.3.1 not_lost() . . . 2299
8.332.3.2 lost_by_writer() . . . 2299
8.332.3.3 lost_by_instances_limit() . . . 2299
8.332.3.4 lost_by_remote_writers_per_instance_limit() . . . 2299
8.332.3.5 lost_by_incomplete_coherent_set() . . . 2300
8.332.3.6 lost_by_large_coherent_set() . . . 2300
8.332.3.7 lost_by_samples_per_remote_writer_limit() . . . 2300
8.332.3.8 lost_by_virtual_writers_limit() . . . 2301
8.332.3.9 lost_by_remote_writers_per_sample_limit() . . . 2301
8.332.3.10 lost_by_availability_waiting_time() . . . 2301
8.332.3.11 lost_by_remote_writers_per_virtual_queue_limit() . . . 2302
8.332.3.12 lost_by_out_of_memory() . . . 2302
8.332.3.13 lost_by_unknown_instance() . . . 2302
8.332.3.14 lost_by_deserialization_failure() . . . 2302
8.332.3.15 lost_by_decode_failure() . . . 2303
8.332.3.16 lost_by_samples_per_instance_limit() . . . 2303
8.332.3.17 lost_by_samples_limit() . . . 2303
8.333 dds::core::status::SampleLostStatus Class Reference . . . 2303
8.333.1 Detailed Description . . . 2304
8.333.2 Member Function Documentation . . . 2304
8.333.2.1 total_count() . . . 2304
8.333.2.2 total_count_change() . . . 2304
8.333.2.3 last_reason() . . . 2304
8.334 rti::sub::SampleProcessor Class Reference . . . 2305
8.334.1 Detailed Description . . . 2305
8.334.2 SampleProcessor Thread Safety . . . 2306
8.334.3 Constructor & Destructor Documentation . . . 2307
8.334.3.1 SampleProcessor() [1/3] . . . 2307
8.334.3.2 SampleProcessor() [2/3] . . . 2308
8.334.3.3 SampleProcessor() [3/3] . . . 2308
8.334.4 Member Function Documentation . . . 2309
8.334.4.1 attach_reader() . . . 2309
8.334.4.2 detach_reader() . . . 2310

8.334.4.3 readers() . . . 2310
8.335 dds::core::status::SampleRejectedState Class Reference . . . 2310
8.335.1 Detailed Description . . . 2311
8.335.2 Constructor & Destructor Documentation . . . 2312
8.335.2.1 SampleRejectedState() [1/2] . . . 2312
8.335.2.2 SampleRejectedState() [2/2] . . . 2312
8.335.3 Member Function Documentation . . . 2312
8.335.3.1 not_rejected() . . . 2312
8.335.3.2 rejected_by_samples_limit() . . . 2313
8.335.3.3 rejected_by_instances_limit() . . . 2313
8.335.3.4 rejected_by_samples_per_instance_limit() . . . 2313
8.335.3.5 rejected_by_samples_per_remote_writer_limit() . . . 2314
8.335.3.6 rejected_by_remote_writers_per_virtual_queue_limit() . . . 2314
8.335.3.7 rejected_by_decode_failure() . . . 2314
8.336 dds::core::status::SampleRejectedStatus Class Reference . . . 2315
8.336.1 Detailed Description . . . 2315
8.336.2 Member Function Documentation . . . 2315
8.336.2.1 total_count() . . . 2315
8.336.2.2 total_count_change() . . . 2315
8.336.2.3 last_reason() . . . 2316
8.336.2.4 last_instance_handle() . . . 2316
8.337 dds::sub::status::SampleState Class Reference . . . 2316
8.337.1 Detailed Description . . . 2317
8.337.2 Member Typedef Documentation . . . 2317
8.337.2.1 MaskType . . . 2317
8.337.3 Constructor & Destructor Documentation . . . 2317
8.337.3.1 SampleState() . . . 2317
8.337.4 Member Function Documentation . . . 2318
8.337.4.1 read() . . . 2318
8.337.4.2 not_read() . . . 2318
8.337.4.3 any() . . . 2318
8.337.5 Friends And Related Function Documentation . . . 2318
8.337.5.1 operator<<() . . . 2319
8.338 rti::config::ScopedLoggerVerbosity Class Reference . . . 2319
8.338.1 Detailed Description . . . 2319
8.338.2 Constructor & Destructor Documentation . . . 2319
8.338.2.1 ScopedLoggerVerbosity() . . . 2319
8.338.2.2 ~ScopedLoggerVerbosity() . . . 2320
8.339 dds::sub::DataReader< T >::Selector Class Reference . . . 2320

8.339.1 Detailed Description . . . 2321
8.339.2 Constructor & Destructor Documentation . . . 2321
8.339.2.1 Selector() . . . 2321
8.339.3 Member Function Documentation . . . 2322
8.339.3.1 instance() . . . 2322
8.339.3.2 next_instance() . . . 2322
8.339.3.3 state() . . . 2323
8.339.3.4 content() . . . 2324
8.339.3.5 condition() . . . 2324
8.339.3.6 max_samples() . . . 2325
8.339.3.7 read() [1/3] . . . 2325
8.339.3.8 take() [1/3] . . . 2326
8.339.3.9 reader() . . . 2326
8.339.3.10 read() [2/3] . . . 2326
8.339.3.11 take() [2/3] . . . 2327
8.339.3.12 read() [3/3] . . . 2327
8.339.3.13 take() [3/3] . . . 2328
8.340 rti::flat::SequenceIterator< E, OffsetKind > Class Template Reference . . . 2328
8.340.1 Detailed Description . . . 2330
8.340.2 Member Typedef Documentation . . . 2331
8.340.2.1 iterator_category . . . 2331
8.340.2.2 value_type . . . 2331
8.340.2.3 reference . . . 2331
8.340.2.4 pointer . . . 2332
8.340.2.5 difference_type . . . 2332
8.340.3 Constructor & Destructor Documentation . . . 2332
8.340.3.1 SequenceIterator() . . . 2332
8.340.4 Member Function Documentation . . . 2332
8.340.4.1 is_null() . . . 2332
8.340.4.2 operator*() . . . 2333
8.340.4.3 operator->() . . . 2333
8.340.4.4 advance() . . . 2333
8.340.4.5 operator++() [1/2] . . . 2333
8.340.4.6 operator++() [2/2] . . . 2334
8.340.5 Friends And Related Function Documentation . . . 2334
8.340.5.1 operator< . . . 2334
8.340.5.2 operator> . . . 2334
8.340.5.3 operator<= . . . 2334
8.340.5.4 operator>= . . . 2335

8.340.5.5 operator== . . . 2335
8.340.5.6 operator"!= . . . 2335
8.341 rti::core::SequenceNumber Class Reference . . . 2335
8.341.1 Detailed Description . . . 2337
8.341.2 Constructor & Destructor Documentation . . . 2337
8.341.2.1 SequenceNumber() [1/3] . . . 2337
8.341.2.2 SequenceNumber() [2/3] . . . 2337
8.341.2.3 SequenceNumber() [3/3] . . . 2338
8.341.3 Member Function Documentation . . . 2338
8.341.3.1 zero() . . . 2338
8.341.3.2 unknown() . . . 2338
8.341.3.3 maximum() . . . 2338
8.341.3.4 automatic() . . . 2339
8.341.3.5 high() [1/2] . . . 2339
8.341.3.6 high() [2/2] . . . 2339
8.341.3.7 low() [1/2] . . . 2339
8.341.3.8 low() [2/2] . . . 2339
8.341.3.9 value() [1/2] . . . 2340
8.341.3.10 value() [2/2] . . . 2340
8.341.3.11 operator+() . . . 2340
8.341.3.12 operator-() . . . 2340
8.341.3.13 operator+=() . . . 2340
8.341.3.14 operator-=() . . . 2341
8.341.3.15 operator++() [1/2] . . . 2341
8.341.3.16 operator++() [2/2] . . . 2341
8.341.3.17 operator--() [1/2] . . . 2341
8.341.3.18 operator--() [2/2] . . . 2341
8.341.3.19 operator<() . . . 2342
8.341.3.20 operator<=() . . . 2342
8.341.3.21 operator>() . . . 2342
8.341.3.22 operator>=() . . . 2342
8.341.4 Friends And Related Function Documentation . . . 2342
8.341.4.1 operator<<() . . . 2342
8.342 rti::flat::SequenceOffset< ElementOffset > Class Template Reference . . . 2343
8.342.1 Detailed Description . . . 2343
8.342.2 Member Function Documentation . . . 2344
8.342.2.1 get_element() . . . 2344
8.342.2.2 element_count() . . . 2344
8.343 dds::core::xtypes::SequenceType Class Reference . . . 2344

8.343.1 Detailed Description . . . 2345
8.343.2 Constructor & Destructor Documentation . . . 2345
8.343.2.1 SequenceType() [1/4] . . . 2345
8.343.2.2 SequenceType() [2/4] . . . 2345
8.343.2.3 SequenceType() [3/4] . . . 2346
8.343.2.4 SequenceType() [4/4] . . . 2346
8.344 dds::rpc::Server Class Reference . . . 2346
8.344.1 Detailed Description . . . 2347
8.344.2 Constructor & Destructor Documentation . . . 2347
8.344.2.1 Server() [1/2] . . . 2347
8.344.2.2 Server() [2/2] . . . 2347
8.344.3 Member Function Documentation . . . 2347
8.344.3.1 run() [1/2] . . . 2347
8.344.3.2 run() [2/2] . . . 2348
8.344.3.3 close() . . . 2348
8.345 dds::rpc::ServerParams Class Reference . . . 2348
8.345.1 Detailed Description . . . 2349
8.345.2 Member Function Documentation . . . 2349
8.345.2.1 thread_pool_size() [1/2] . . . 2349
8.345.2.2 thread_pool_size() [2/2] . . . 2349
8.345.2.3 async_waitset_property() [1/2] . . . 2350
8.345.2.4 async_waitset_property() [2/2] . . . 2350
8.346 rti::core::policy::Service Class Reference . . . 2350
8.346.1 Detailed Description . . . 2351
8.346.2 Constructor & Destructor Documentation . . . 2351
8.346.2.1 Service() [1/2] . . . 2352
8.346.2.2 Service() [2/2] . . . 2352
8.346.3 Member Function Documentation . . . 2352
8.346.3.1 NoService() . . . 2352
8.346.3.2 PersistenceService() . . . 2352
8.346.3.3 QueuingService() . . . 2352
8.346.3.4 RoutingService() . . . 2353
8.346.3.5 RecordingService() . . . 2353
8.346.3.6 ReplayService() . . . 2353
8.346.3.7 DatabaseIntegrationService() . . . 2353
8.346.3.8 WebIntegrationService() . . . 2353
8.346.3.9 ObservabilityCollectorService() . . . 2353
8.346.3.10 AdminConsoleService() . . . 2354
8.346.3.11 kind() [1/2] . . . 2354

8.346.3.12 kind() [2/2] . . . 2354
8.347 dds::rpc::ServiceEndpoint< Dispatcher > Class Template Reference . . . 2354
8.347.1 Detailed Description . . . 2355
8.347.2 Member Typedef Documentation . . . 2355
8.347.2.1 InterfaceType . . . 2355
8.347.2.2 RequestType . . . 2356
8.347.2.3 ReplyType . . . 2356
8.347.3 Constructor & Destructor Documentation . . . 2356
8.347.3.1 ServiceEndpoint() . . . 2356
8.347.4 Member Function Documentation . . . 2356
8.347.4.1 close() . . . 2356
8.347.4.2 request_datareader() . . . 2357
8.347.4.3 reply_datawriter() . . . 2357
8.348 rti::core::policy::ServiceKind_def Struct Reference . . . 2357
8.348.1 Detailed Description . . . 2358
8.348.2 Member Enumeration Documentation . . . 2358
8.348.2.1 type . . . 2358
8.349 rti::topic::ServiceRequest Class Reference . . . 2358
8.349.1 Detailed Description . . . 2359
8.349.2 Member Function Documentation . . . 2359
8.349.2.1 service_id() . . . 2359
8.349.2.2 instance_id() . . . 2359
8.349.2.3 request_body() . . . 2359
8.350 rti::core::status::ServiceRequestAcceptedStatus Class Reference . . . 2360
8.350.1 Detailed Description . . . 2360
8.350.2 Member Function Documentation . . . 2360
8.350.2.1 total_count() . . . 2360
8.350.2.2 current_count() . . . 2361
8.350.2.3 last_request_handle() . . . 2361
8.350.2.4 service_id() . . . 2361
8.351 rti::core::ServiceRequestId_def Struct Reference . . . 2361
8.351.1 Detailed Description . . . 2362
8.351.2 Member Enumeration Documentation . . . 2362
8.351.2.1 type . . . 2362
8.352 dds::sub::SharedSamples< T, DELEGATE > Class Template Reference . . . 2362
8.352.1 Detailed Description . . . 2363
8.352.2 Constructor & Destructor Documentation . . . 2363
8.352.2.1 SharedSamples() . . . 2363
8.352.3 Member Function Documentation . . . 2364

8.352.3.1 begin() . . . 2364
8.352.3.2 end() . . . 2364
8.352.3.3 operator[]() . . . 2364
8.352.3.4 length() . . . 2365
8.352.4 Friends And Related Function Documentation . . . 2365
8.352.4.1 unpack() [1/3] . . . 2365
8.352.4.2 unpack() [2/3] . . . 2366
8.352.4.3 unpack() [3/3] . . . 2366
8.353 rti::sub::SharedSamples< T > Class Template Reference . . . 2367
8.353.1 Detailed Description . . . 2367
8.353.2 Member Typedef Documentation . . . 2367
8.353.2.1 iterator . . . 2367
8.354 rti::request::SimpleReplier< RequestType, ReplyType > Class Template Reference . . . 2368
8.354.1 Detailed Description . . . 2368
8.354.2 Constructor & Destructor Documentation . . . 2369
8.354.2.1 SimpleReplier() [1/2] . . . 2369
8.354.2.2 SimpleReplier() [2/2] . . . 2369
8.355 rti::util::heap_monitoring::SnapshotContentFormat_def Struct Reference . . . 2370
8.355.1 Detailed Description . . . 2370
8.355.2 Member Enumeration Documentation . . . 2370
8.355.2.1 type . . . 2370
8.356 rti::util::heap_monitoring::SnapshotOutputFormat_def Struct Reference . . . 2371
8.356.1 Detailed Description . . . 2371
8.356.2 Member Enumeration Documentation . . . 2371
8.356.2.1 type . . . 2371
8.357 dds::core::cond::StatusCondition Class Reference . . . 2372
8.357.1 Detailed Description . . . 2372
8.357.2 Constructor & Destructor Documentation . . . 2373
8.357.2.1 StatusCondition() . . . 2373
8.357.3 Member Function Documentation . . . 2373
8.357.3.1 enabled_statuses() [1/2] . . . 2374
8.357.3.2 enabled_statuses() [2/2] . . . 2374
8.357.3.3 entity() . . . 2374
8.357.3.4 handler() . . . 2375
8.357.3.5 reset_handler() . . . 2375
8.358 dds::core::status::StatusMask Class Reference . . . 2375
8.358.1 Detailed Description . . . 2377
8.358.2 Member Typedef Documentation . . . 2378
8.358.2.1 MaskType . . . 2378

8.358.3 Constructor & Destructor Documentation . . . 2378
8.358.3.1 StatusMask() [1/3] . . . 2378
8.358.3.2 StatusMask() [2/3] . . . 2378
8.358.3.3 StatusMask() [3/3] . . . 2379
8.358.4 Member Function Documentation . . . 2379
8.358.4.1 all() . . . 2379
8.358.4.2 none() . . . 2379
8.358.4.3 inconsistent_topic() . . . 2379
8.358.4.4 offered_deadline_missed() . . . 2380
8.358.4.5 requested_deadline_missed() . . . 2380
8.358.4.6 offered_incompatible_qos() . . . 2381
8.358.4.7 requested_incompatible_qos() . . . 2381
8.358.4.8 sample_lost() . . . 2382
8.358.4.9 sample_rejected() . . . 2382
8.358.4.10 data_on_readers() . . . 2383
8.358.4.11 data_available() . . . 2383
8.358.4.12 liveliness_lost() . . . 2383
8.358.4.13 liveliness_changed() . . . 2384
8.358.4.14 publication_matched() . . . 2384
8.358.4.15 subscription_matched() . . . 2385
8.358.4.16 reliable_writer_cache_changed() . . . 2385
8.358.4.17 reliable_reader_activity_changed() . . . 2386
8.358.4.18 datawriter_cache() . . . 2386
8.358.4.19 datawriter_protocol() . . . 2386
8.358.4.20 datareader_cache() . . . 2387
8.358.4.21 datareader_protocol() . . . 2387
8.358.4.22 datawriter_application_acknowledgment() . . . 2387
8.358.4.23 datawriter_instance_replaced() . . . 2388
8.358.4.24 service_request_accepted() . . . 2388
8.358.4.25 invalid_local_advance_notice() . . . 2389
8.358.4.26 sample_removed() . . . 2389
8.358.5 Friends And Related Function Documentation . . . 2389
8.358.5.1 get_status() . . . 2390
8.359 rti::util::StreamFlagSaver Class Reference . . . 2390
8.359.1 Detailed Description . . . 2390
8.360 rti::sub::status::StreamKind Class Reference . . . 2390
8.360.1 Detailed Description . . . 2391
8.360.2 Member Typedef Documentation . . . 2391
8.360.2.1 MaskType . . . 2391

8.360.3 Constructor & Destructor Documentation . . . 2391
8.360.3.1 StreamKind() . . . 2391
8.360.4 Member Function Documentation . . . 2392
8.360.4.1 live() . . . 2392
8.360.4.2 topic_query() . . . 2392
8.360.4.3 any() . . . 2392
8.361 rti::flat::StringBuilder Class Reference . . . 2392
8.361.1 Detailed Description . . . 2393
8.361.2 Member Function Documentation . . . 2393
8.361.2.1 set_string() . . . 2393
8.361.2.2 finish() . . . 2394
8.362 rti::flat::StringOffset Class Reference . . . 2394
8.362.1 Detailed Description . . . 2395
8.362.2 Member Function Documentation . . . 2395
8.362.2.1 get_string() [1/2] . . . 2395
8.362.2.2 get_string() [2/2] . . . 2395
8.362.2.3 element_count() . . . 2395
8.363 dds::core::StringTopicType Class Reference . . . 2395
8.363.1 Detailed Description . . . 2396
8.363.2 Constructor & Destructor Documentation . . . 2396
8.363.2.1 StringTopicType() [1/4] . . . 2396
8.363.2.2 StringTopicType() [2/4] . . . 2397
8.363.2.3 StringTopicType() [3/4] . . . 2397
8.363.2.4 StringTopicType() [4/4] . . . 2397
8.363.3 Member Function Documentation . . . 2397
8.363.3.1 operator std::string() . . . 2398
8.363.3.2 operator dds::core::string &() . . . 2398
8.363.3.3 operator const dds::core::string &() . . . 2398
8.363.3.4 data() [1/3] . . . 2398
8.363.3.5 data() [2/3] . . . 2399
8.363.3.6 data() [3/3] . . . 2399
8.364 dds::core::xtypes::StringType Class Reference . . . 2399
8.364.1 Detailed Description . . . 2400
8.364.2 Constructor & Destructor Documentation . . . 2400
8.364.2.1 StringType() . . . 2400
8.365 dds::core::xtypes::StructType Class Reference . . . 2400
8.365.1 Detailed Description . . . 2402
8.365.2 Constructor & Destructor Documentation . . . 2402
8.365.2.1 StructType() [1/12] . . . 2402

8.365.2.2 StructType() [2/12] . . . 2402
8.365.2.3 StructType() [3/12] . . . 2403
8.365.2.4 StructType() [4/12] . . . 2403
8.365.2.5 StructType() [5/12] . . . 2404
8.365.2.6 StructType() [6/12] . . . 2404
8.365.2.7 StructType() [7/12] . . . 2406
8.365.2.8 StructType() [8/12] . . . 2406
8.365.2.9 StructType() [9/12] . . . 2406
8.365.2.10 StructType() [10/12] . . . 2407
8.365.2.11 StructType() [11/12] . . . 2407
8.365.2.12 StructType() [12/12] . . . 2407
8.365.3 Member Function Documentation . . . 2408
8.365.3.1 extensibility_kind() . . . 2408
8.365.3.2 has_parent() . . . 2408
8.365.3.3 parent() . . . 2408
8.365.3.4 find_member_by_id() . . . 2408
8.365.3.5 add_member() [1/2] . . . 2409
8.365.3.6 add_members() [1/3] . . . 2409
8.365.3.7 add_members() [2/3] . . . 2409
8.365.3.8 add_members() [3/3] . . . 2409
8.365.3.9 add_member() [2/2] . . . 2409
8.365.4 Friends And Related Function Documentation . . . 2410
8.365.4.1 create_type_from_tuple() . . . 2410
8.366 dds::sub::Subscriber Class Reference . . . 2410
8.366.1 Detailed Description . . . 2412
8.366.2 Constructor & Destructor Documentation . . . 2413
8.366.2.1 Subscriber() [1/3] . . . 2413
8.366.2.2 Subscriber() [2/3] . . . 2413
8.366.2.3 Subscriber() [3/3] . . . 2414
8.366.3 Member Function Documentation . . . 2414
8.366.3.1 notify_datareaders() . . . 2414
8.366.3.2 listener() [1/2] . . . 2415
8.366.3.3 listener() [2/2] . . . 2415
8.366.3.4 set_listener() [1/2] . . . 2416
8.366.3.5 set_listener() [2/2] . . . 2416
8.366.3.6 get_listener() . . . 2416
8.366.3.7 qos() [1/2] . . . 2417
8.366.3.8 qos() [2/2] . . . 2417
8.366.3.9 default_datareader_qos() [1/2] . . . 2417

8.366.3.10 default_datareader_qos() [2/2] . . . 2417
8.366.3.11 participant() . . . 2418
8.366.3.12 close_contained_entities() . . . 2418
8.366.4 Friends And Related Function Documentation . . . 2419
8.366.4.1 builtin_subscriber() . . . 2419
8.366.4.2 find_subscribers() [1/2] . . . 2420
8.366.4.3 find_subscribers() [2/2] . . . 2420
8.366.4.4 find_subscriber() . . . 2421
8.366.4.5 implicit_subscriber() . . . 2422
8.367 dds::sub::SubscriberListener Class Reference . . . 2422
8.367.1 Detailed Description . . . 2423
8.368 dds::sub::qos::SubscriberQos Class Reference . . . 2423
8.368.1 Detailed Description . . . 2424
8.368.2 Member Function Documentation . . . 2425
8.368.2.1 policy() [1/2] . . . 2425
8.368.2.2 policy() [2/2] . . . 2425
8.368.3 Friends And Related Function Documentation . . . 2425
8.368.3.1 to_string() [1/3] . . . 2426
8.368.3.2 to_string() [2/3] . . . 2426
8.368.3.3 to_string() [3/3] . . . 2427
8.368.3.4 operator<<() . . . 2427
8.369 dds::topic::SubscriptionBuiltinTopicData Class Reference . . . 2428
8.369.1 Detailed Description . . . 2429
8.369.2 Member Function Documentation . . . 2430
8.369.2.1 TSubscriptionBuiltinTopicData() . . . 2430
8.369.2.2 key() . . . 2430
8.369.2.3 participant_key() . . . 2430
8.369.2.4 topic_name() . . . 2430
8.369.2.5 type_name() . . . 2431
8.369.2.6 durability() . . . 2431
8.369.2.7 deadline() . . . 2431
8.369.2.8 latency_budget() . . . 2431
8.369.2.9 liveliness() . . . 2432
8.369.2.10 reliability() . . . 2432
8.369.2.11 ownership() . . . 2432
8.369.2.12 destination_order() . . . 2432
8.369.2.13 user_data() . . . 2433
8.369.2.14 time_based_filter() . . . 2433
8.369.2.15 presentation() . . . 2433

8.369.2.16 partition() . . . 2433
8.369.2.17 topic_data() . . . 2434
8.369.2.18 group_data() . . . 2434
8.369.2.19 representation() . . . 2434
8.369.2.20 data_tag() . . . 2434
8.369.2.21 type() . . . 2435
8.369.2.22 get_type_no_copy() . . . 2435
8.369.2.23 subscriber_key() . . . 2435
8.369.2.24 property() . . . 2436
8.369.2.25 unicast_locators() . . . 2436
8.369.2.26 multicast_locators() . . . 2436
8.369.2.27 content_filter_property() . . . 2436
8.369.2.28 virtual_guid() . . . 2436
8.369.2.29 rtps_protocol_version() . . . 2437
8.369.2.30 rtps_vendor_id() . . . 2437
8.369.2.31 product_version() . . . 2437
8.369.2.32 disable_positive_acks() . . . 2437
8.369.2.33 subscription_name() . . . 2437
8.369.2.34 trust_protection_info() . . . 2438
8.369.2.35 trust_algorithm_info() . . . 2438
8.369.2.36 related_datawriter_key() . . . 2438
8.369.2.37 service() . . . 2438
8.370 dds::core::status::SubscriptionMatchedStatus Class Reference . . . 2439
8.370.1 Detailed Description . . . 2439
8.370.2 Member Function Documentation . . . 2440
8.370.2.1 total_count() . . . 2440
8.370.2.2 total_count_change() . . . 2440
8.370.2.3 current_count() . . . 2440
8.370.2.4 current_count_change() . . . 2440
8.370.2.5 last_publication_handle() . . . 2441
8.370.2.6 current_count_peak() . . . 2441
8.371 dds::pub::SuspendedPublication Class Reference . . . 2441
8.371.1 Detailed Description . . . 2442
8.371.2 Constructor & Destructor Documentation . . . 2442
8.371.2.1 SuspendedPublication() . . . 2442
8.371.2.2 ∼SuspendedPublication() . . . 2443
8.371.3 Member Function Documentation . . . 2443
8.371.3.1 resume() . . . 2443
8.372 rti::core::policy::SystemResourceLimits Class Reference . . . 2443

8.372.1 Detailed Description . . . 2444
8.372.2 Usage . . . 2444
8.372.3 Constructor & Destructor Documentation . . . 2444
8.372.3.1 SystemResourceLimits() [1/3] . . . 2444
8.372.3.2 SystemResourceLimits() [2/3] . . . 2444
8.372.3.3 SystemResourceLimits() [3/3] . . . 2445
8.372.4 Member Function Documentation . . . 2445
8.372.4.1 max_objects_per_thread() [1/2] . . . 2445
8.372.4.2 initial_objects_per_thread() [1/2] . . . 2446
8.372.4.3 max_objects_per_thread() [2/2] . . . 2446
8.372.4.4 initial_objects_per_thread() [2/2] . . . 2446
8.373 dds::core::TEntityQos< DELEGATE > Class Template Reference . . . 2446
8.373.1 Detailed Description . . . 2447
8.373.2 Member Function Documentation . . . 2447
8.373.2.1 policy() . . . 2447
8.373.2.2 operator<<() . . . 2448
8.373.2.3 operator>>() . . . 2448
8.373.2.4 operator=() . . . 2448
8.374 rti::core::ThreadSettings Class Reference . . . 2449
8.374.1 Detailed Description . . . 2449
8.374.2 Constructor & Destructor Documentation . . . 2450
8.374.2.1 ThreadSettings() [1/2] . . . 2450
8.374.2.2 ThreadSettings() [2/2] . . . 2450
8.374.3 Member Function Documentation . . . 2450
8.374.3.1 mask() [1/2] . . . 2450
8.374.3.2 mask() [2/2] . . . 2450
8.374.3.3 priority() [1/2] . . . 2451
8.374.3.4 priority() [2/2] . . . 2451
8.374.3.5 stack_size() [1/2] . . . 2451
8.374.3.6 stack_size() [2/2] . . . 2451
8.374.3.7 cpu_list() [1/2] . . . 2451
8.374.3.8 cpu_list() [2/2] . . . 2452
8.374.3.9 cpu_rotation() [1/2] . . . 2452
8.374.3.10 cpu_rotation() [2/2] . . . 2452
8.375 rti::core::ThreadSettingsCpuRotationKind_def Struct Reference . . . 2453
8.375.1 Detailed Description . . . 2453
8.375.2 Controlling CPU Core Affinity for RTI Threads . . . 2453
8.375.3 Member Enumeration Documentation . . . 2453
8.375.3.1 type . . . 2453

8.376 rti::core::ThreadSettingsKindMask Class Reference 2454
8.376.1 Detailed Description 2455
8.376.2 Member Typedef Documentation 2455
8.376.2.1 MaskType 2455
8.376.3 Constructor & Destructor Documentation 2455
8.376.3.1 ThreadSettingsKindMask() [1/3] 2455
8.376.3.2 ThreadSettingsKindMask() [2/3] 2455
8.376.3.3 ThreadSettingsKindMask() [3/3] 2455
8.376.4 Member Function Documentation 2456
8.376.4.1 floating_point() 2456
8.376.4.2 stdio() 2456
8.376.4.3 realtime_priority() 2456
8.376.4.4 priority_enforce() 2456
8.376.4.5 cancel_asynchronous() 2456
8.377 dds::core::Time Class Reference 2457
8.377.1 Detailed Description 2458
8.377.2 Constructor & Destructor Documentation 2458
8.377.2.1 Time() [1/2] 2458
8.377.2.2 Time() [2/2] 2459
8.377.3 Member Function Documentation 2459
8.377.3.1 invalid() 2459
8.377.3.2 zero() 2459
8.377.3.3 maximum() 2459
8.377.3.4 from_microsecs() 2459
8.377.3.5 from_millisecs() 2460
8.377.3.6 from_secs() 2460
8.377.3.7 sec() [1/2] 2461
8.377.3.8 sec() [2/2] 2461
8.377.3.9 nanosec() [1/2] 2461
8.377.3.10 nanosec() [2/2] 2461
8.377.3.11 compare() 2462
8.377.3.12 operator>() 2462
8.377.3.13 operator>=() 2462
8.377.3.14 operator==() 2464
8.377.3.15 operator"!=() 2464
8.377.3.16 operator<=() 2465
8.377.3.17 operator<() 2465
8.377.3.18 operator+=() 2465
8.377.3.19 operator-=() 2466

8.377.3.20 to_millisecs() . . . 2466
8.377.3.21 to_microsecs() . . . 2466
8.377.3.22 to_nanosecs() . . . 2467
8.377.3.23 to_secs() . . . 2467
8.377.4 Friends And Related Function Documentation . . . 2467
8.377.4.1 operator+() [1/2] . . . 2467
8.377.4.2 operator+() [2/2] . . . 2468
8.377.4.3 operator-() [1/2] . . . 2468
8.377.4.4 operator-() [2/2] . . . 2469
8.378 dds::core::policy::TimeBasedFilter Class Reference . . . 2469
8.378.1 Detailed Description . . . 2470
8.378.2 Usage . . . 2470
8.378.3 Consistency . . . 2471
8.378.4 Constructor & Destructor Documentation . . . 2471
8.378.4.1 TimeBasedFilter() [1/2] . . . 2472
8.378.4.2 TimeBasedFilter() [2/2] . . . 2472
8.378.5 Member Function Documentation . . . 2472
8.378.5.1 minimum_separation() [1/2] . . . 2472
8.378.5.2 minimum_separation() [2/2] . . . 2472
8.379 dds::core::TimeoutError Class Reference . . . 2473
8.379.1 Detailed Description . . . 2473
8.379.2 Member Function Documentation . . . 2473
8.379.2.1 what() . . . 2473
8.380 dds::topic::Topic< T > Class Template Reference . . . 2474
8.380.1 Detailed Description . . . 2476
8.380.2 Constructor & Destructor Documentation . . . 2477
8.380.2.1 Topic() [1/9] . . . 2477
8.380.2.2 Topic() [2/9] . . . 2477
8.380.2.3 Topic() [3/9] . . . 2478
8.380.2.4 Topic() [4/9] . . . 2478
8.380.2.5 Topic() [5/9] . . . 2479
8.380.2.6 Topic() [6/9] . . . 2479
8.380.2.7 Topic() [7/9] . . . 2480
8.380.2.8 Topic() [8/9] . . . 2480
8.380.2.9 Topic() [9/9] . . . 2481
8.380.3 Member Function Documentation . . . 2482
8.380.3.1 listener() [1/2] . . . 2482
8.380.3.2 listener() [2/2] . . . 2482
8.380.3.3 set_listener() [1/2] . . . 2482

8.380.3.4 set_listener() [2/2] . . . 2483
8.380.3.5 get_listener() . . . 2483
8.380.3.6 qos() [1/2] . . . 2484
8.380.3.7 qos() [2/2] . . . 2484
8.380.3.8 inconsistent_topic_status() . . . 2484
8.380.4 Friends And Related Function Documentation . . . 2485
8.380.4.1 discover_any_topic() [1/2] . . . 2485
8.380.4.2 discover_any_topic() [2/2] . . . 2485
8.380.4.3 discover_topic_data() [1/4] . . . 2486
8.380.4.4 discover_topic_data() [2/4] . . . 2486
8.380.4.5 discover_topic_data() [3/4] . . . 2487
8.380.4.6 discover_topic_data() [4/4] . . . 2487
8.380.4.7 ignore() [1/2] . . . 2487
8.380.4.8 ignore() [2/2] . . . 2489
8.380.4.9 find() . . . 2489
8.380.4.10 find_topics() [1/2] . . . 2490
8.380.4.11 find_topics() [2/2] . . . 2491
8.381 rti::topic::topic_type_disabled_copy< TopicType > Struct Template Reference . . . 2492
8.381.1 Detailed Description . . . 2492
8.382 rti::topic::topic_type_has_external_members< TopicType > Struct Template Reference . . . 2492
8.382.1 Detailed Description . . . 2492
8.383 dds::topic::topic_type_name< T > Struct Template Reference . . . 2493
8.383.1 Detailed Description . . . 2493
8.384 dds::topic::topic_type_support< T > Struct Template Reference . . . 2493
8.384.1 Detailed Description . . . 2493
8.385 dds::topic::TopicBuiltinTopicData Class Reference . . . 2493
8.385.1 Detailed Description . . . 2494
8.385.2 Member Function Documentation . . . 2495
8.385.2.1 key() . . . 2495
8.385.2.2 name() . . . 2495
8.385.2.3 type_name() . . . 2495
8.385.2.4 durability() . . . 2496
8.385.2.5 durability_service() . . . 2496
8.385.2.6 deadline() . . . 2496
8.385.2.7 latency_budget() . . . 2497
8.385.2.8 liveliness() . . . 2497
8.385.2.9 reliability() . . . 2497
8.385.2.10 transport_priority() . . . 2498
8.385.2.11 lifespan() . . . 2498

8.385.2.12 destination_order() . . . 2498
8.385.2.13 history() . . . 2499
8.385.2.14 resource_limits() . . . 2499
8.385.2.15 ownership() . . . 2499
8.385.2.16 topic_data() . . . 2500
8.385.2.17 representation() . . . 2500
8.386 dds::core::policy::TopicData Class Reference . . . 2500
8.386.1 Detailed Description . . . 2501
8.386.2 Usage . . . 2501
8.386.3 Constructor & Destructor Documentation . . . 2501
8.386.3.1 TopicData() [1/3] . . . 2501
8.386.3.2 TopicData() [2/3] . . . 2502
8.386.3.3 TopicData() [3/3] . . . 2502
8.386.4 Member Function Documentation . . . 2502
8.386.4.1 value() [1/3] . . . 2502
8.386.4.2 value() [2/3] . . . 2502
8.386.4.3 value() [3/3] . . . 2502
8.386.4.4 begin() . . . 2503
8.386.4.5 end() . . . 2503
8.387 dds::topic::TopicDescription< T > Class Template Reference . . . 2503
8.387.1 Detailed Description . . . 2504
8.387.2 Member Function Documentation . . . 2504
8.387.2.1 name() . . . 2504
8.387.2.2 type_name() . . . 2504
8.387.2.3 participant() . . . 2505
8.388 dds::topic::TopicInstance< T > Class Template Reference . . . 2505
8.388.1 Detailed Description . . . 2505
8.388.2 Constructor & Destructor Documentation . . . 2506
8.388.2.1 TopicInstance() [1/3] . . . 2506
8.388.2.2 TopicInstance() [2/3] . . . 2506
8.388.2.3 TopicInstance() [3/3] . . . 2506
8.388.3 Member Function Documentation . . . 2506
8.388.3.1 operator dds::core::InstanceHandle() . . . 2507
8.388.3.2 handle() [1/2] . . . 2507
8.388.3.3 handle() [2/2] . . . 2507
8.388.3.4 sample() [1/3] . . . 2507
8.388.3.5 sample() [2/3] . . . 2507
8.388.3.6 sample() [3/3] . . . 2507
8.389 dds::topic::TopicListener< T > Class Template Reference . . . 2508

8.389.1 Detailed Description . . . 2508
8.389.2 Member Function Documentation . . . 2509
8.389.2.1 on_inconsistent_topic() . . . 2509
8.390 dds::topic::qos::TopicQos Class Reference . . . 2509
8.390.1 Detailed Description . . . 2510
8.390.2 TopicQos Policies . . . 2510
8.390.3 Constructor & Destructor Documentation . . . 2511
8.390.3.1 TopicQos() . . . 2511
8.390.4 Member Function Documentation . . . 2511
8.390.4.1 policy() [1/3] . . . 2511
8.390.4.2 policy() [2/3] . . . 2512
8.390.4.3 policy() [3/3] . . . 2512
8.390.4.4 operator<<() . . . 2513
8.390.4.5 operator>>() . . . 2513
8.390.5 Friends And Related Function Documentation . . . 2513
8.390.5.1 to_string() [1/3] . . . 2513
8.390.5.2 to_string() [2/3] . . . 2514
8.390.5.3 to_string() [3/3] . . . 2515
8.390.5.4 operator<<() . . . 2515
8.391 rti::sub::TopicQuery Class Reference . . . 2516
8.391.1 Detailed Description . . . 2517
8.391.2 Constructor & Destructor Documentation . . . 2517
8.391.2.1 TopicQuery() . . . 2517
8.391.3 Member Function Documentation . . . 2518
8.391.3.1 close() . . . 2518
8.391.3.2 guid() . . . 2518
8.391.3.3 closed() . . . 2518
8.391.3.4 datareader() . . . 2518
8.391.3.5 UseReaderContentFilter() . . . 2519
8.391.3.6 SelectAll() . . . 2519
8.391.4 Friends And Related Function Documentation . . . 2519
8.391.4.1 find_topic_query() . . . 2519
8.392 rti::sub::TopicQueryData Class Reference . . . 2520
8.392.1 Detailed Description . . . 2520
8.392.2 Member Function Documentation . . . 2520
8.392.2.1 selection() . . . 2521
8.392.2.2 topic_name() . . . 2521
8.392.2.3 original_related_reader_guid() . . . 2521
8.392.3 Friends And Related Function Documentation . . . 2521

8.392.3.1 create_topic_query_data_from_service_request() . . . 2521
8.393 rti::core::policy::TopicQueryDispatch Class Reference . . . 2522
8.393.1 Detailed Description . . . 2523
8.393.2 Constructor & Destructor Documentation . . . 2524
8.393.2.1 TopicQueryDispatch() [1/2] . . . 2524
8.393.2.2 TopicQueryDispatch() [2/2] . . . 2524
8.393.3 Member Function Documentation . . . 2524
8.393.3.1 enable() [1/2] . . . 2524
8.393.3.2 enable() [2/2] . . . 2525
8.393.3.3 publication_period() [1/2] . . . 2525
8.393.3.4 publication_period() [2/2] . . . 2525
8.393.3.5 samples_per_period() [1/2] . . . 2525
8.393.3.6 samples_per_period() [2/2] . . . 2526
8.394 rti::sub::TopicQuerySelection Class Reference . . . 2526
8.394.1 Detailed Description . . . 2526
8.394.2 Member Typedef Documentation . . . 2526
8.394.2.1 Kind . . . 2527
8.394.3 Constructor & Destructor Documentation . . . 2527
8.394.3.1 TopicQuerySelection() [1/2] . . . 2527
8.394.3.2 TopicQuerySelection() [2/2] . . . 2527
8.394.4 Member Function Documentation . . . 2528
8.394.4.1 filter() [1/2] . . . 2528
8.394.4.2 filter() [2/2] . . . 2528
8.394.4.3 kind() [1/2] . . . 2528
8.394.4.4 kind() [2/2] . . . 2528
8.395 rti::sub::TopicQuerySelectionKind_def Struct Reference . . . 2528
8.395.1 Detailed Description . . . 2529
8.395.2 Member Enumeration Documentation . . . 2529
8.395.2.1 type . . . 2529
8.396 rti::util::network_capture::TrafficKindMask Class Reference . . . 2529
8.396.1 Detailed Description . . . 2530
8.396.2 Member Typedef Documentation . . . 2530
8.396.2.1 MaskType . . . 2530
8.396.3 Constructor & Destructor Documentation . . . 2531
8.396.3.1 TrafficKindMask() [1/3] . . . 2531
8.396.3.2 TrafficKindMask() [2/3] . . . 2531
8.396.3.3 TrafficKindMask() [3/3] . . . 2531
8.396.4 Member Function Documentation . . . 2531
8.396.4.1 out() . . . 2532

8.396.4.2 in() . . . 2532
8.396.4.3 default_mask() . . . 2532
8.396.4.4 none() . . . 2532
8.396.4.5 all() . . . 2532
8.397 TransportAllocationSettings_t Struct Reference . . . 2533
8.397.1 Detailed Description . . . 2533
8.398 rti::core::policy::TransportBuiltin Class Reference . . . 2533
8.398.1 Detailed Description . . . 2534
8.398.2 Constructor & Destructor Documentation . . . 2535
8.398.2.1 TransportBuiltin() [1/2] . . . 2535
8.398.2.2 TransportBuiltin() [2/2] . . . 2535
8.398.3 Member Function Documentation . . . 2535
8.398.3.1 All() . . . 2535
8.398.3.2 None() . . . 2535
8.398.3.3 Shmem() . . . 2535
8.398.3.4 UDPv4() . . . 2536
8.398.3.5 UDPv6() . . . 2536
8.398.3.6 UDPv4_WAN() . . . 2536
8.398.3.7 mask() [1/2] . . . 2536
8.398.3.8 mask() [2/2] . . . 2536
8.398.4 Member Data Documentation . . . 2536
8.398.4.1 SHMEM_ALIAS . . . 2537
8.398.4.2 UDPv4_ALIAS . . . 2537
8.398.4.3 UDPv4_WAN_ALIAS . . . 2537
8.398.4.4 UDPv6_ALIAS . . . 2537
8.399 rti::core::policy::TransportBuiltinMask Class Reference . . . 2537
8.399.1 Detailed Description . . . 2538
8.399.2 Member Typedef Documentation . . . 2538
8.399.2.1 MaskType . . . 2538
8.399.3 Constructor & Destructor Documentation . . . 2538
8.399.3.1 TransportBuiltinMask() [1/2] . . . 2539
8.399.3.2 TransportBuiltinMask() [2/2] . . . 2539
8.399.4 Member Function Documentation . . . 2539
8.399.4.1 all() . . . 2539
8.399.4.2 none() . . . 2539
8.399.4.3 shmem() . . . 2539
8.399.4.4 udpv4() . . . 2540
8.399.4.5 udpv4_wan() . . . 2540
8.399.4.6 udpv6() . . . 2540

8.400 rti::core::TransportClassId_def Struct Reference . . . 2540
8.400.1 Detailed Description . . . 2541
8.400.2 Member Enumeration Documentation . . . 2541
8.400.2.1 type . . . 2541
8.401 rti::core::TransportInfo Class Reference . . . 2541
8.401.1 Detailed Description . . . 2542
8.401.2 Constructor & Destructor Documentation . . . 2542
8.401.2.1 TransportInfo() [1/2] . . . 2542
8.401.2.2 TransportInfo() [2/2] . . . 2542
8.401.3 Member Function Documentation . . . 2542
8.401.3.1 class_id() [1/2] . . . 2543
8.401.3.2 class_id() [2/2] . . . 2543
8.401.3.3 message_size_max() [1/2] . . . 2543
8.401.3.4 message_size_max() [2/2] . . . 2543
8.402 rti::core::policy::TransportMulticast Class Reference . . . 2543
8.402.1 Detailed Description . . . 2544
8.402.2 Constructor & Destructor Documentation . . . 2544
8.402.2.1 TransportMulticast() [1/2] . . . 2545
8.402.2.2 TransportMulticast() [2/2] . . . 2545
8.402.3 Member Function Documentation . . . 2545
8.402.3.1 settings() [1/2] . . . 2545
8.402.3.2 settings() [2/2] . . . 2546
8.402.3.3 kind() [1/2] . . . 2546
8.402.3.4 kind() [2/2] . . . 2546
8.403 rti::core::policy::TransportMulticastKind_def Struct Reference . . . 2547
8.403.1 Detailed Description . . . 2547
8.403.2 Member Enumeration Documentation . . . 2547
8.403.2.1 type . . . 2547
8.404 rti::core::policy::TransportMulticastMapping Class Reference . . . 2547
8.404.1 Detailed Description . . . 2548
8.404.2 Member Function Documentation . . . 2549
8.404.2.1 mappings() [1/2] . . . 2549
8.404.2.2 mappings() [2/2] . . . 2549
8.405 rti::core::TransportMulticastSettings Class Reference . . . 2549
8.405.1 Detailed Description . . . 2550
8.405.2 Constructor & Destructor Documentation . . . 2550
8.405.2.1 TransportMulticastSettings() . . . 2550
8.405.3 Member Function Documentation . . . 2550
8.405.3.1 transports() [1/2] . . . 2550

8.405.3.2 transports() [2/2] . . . 2551
8.405.3.3 receive_address() [1/2] . . . 2551
8.405.3.4 receive_address() [2/2] . . . 2551
8.405.3.5 receive_port() [1/2] . . . 2551
8.405.3.6 receive_port() [2/2] . . . 2552
8.406 dds::core::policy::TransportPriority Class Reference . . . 2552
8.406.1 Detailed Description . . . 2552
8.406.2 Usage . . . 2553
8.406.3 Constructor & Destructor Documentation . . . 2553
8.406.3.1 TransportPriority() [1/2] . . . 2553
8.406.3.2 TransportPriority() [2/2] . . . 2553
8.406.4 Member Function Documentation . . . 2553
8.406.4.1 value() [1/2] . . . 2554
8.406.4.2 value() [2/2] . . . 2554
8.407 rti::core::policy::TransportSelection Class Reference . . . 2554
8.407.1 Detailed Description . . . 2555
8.407.2 Constructor & Destructor Documentation . . . 2555
8.407.2.1 TransportSelection() [1/2] . . . 2555
8.407.2.2 TransportSelection() [2/2] . . . 2555
8.407.3 Member Function Documentation . . . 2555
8.407.3.1 enabled_transports() [1/2] . . . 2556
8.407.3.2 enabled_transports() [2/2] . . . 2556
8.408 rti::core::policy::TransportUnicast Class Reference . . . 2556
8.408.1 Detailed Description . . . 2557
8.408.2 Usage . . . 2557
8.408.3 Constructor & Destructor Documentation . . . 2558
8.408.3.1 TransportUnicast() [1/2] . . . 2558
8.408.3.2 TransportUnicast() [2/2] . . . 2558
8.408.4 Member Function Documentation . . . 2558
8.408.4.1 settings() [1/2] . . . 2559
8.408.4.2 settings() [2/2] . . . 2559
8.409 rti::core::TransportUnicastSettings Class Reference . . . 2559
8.409.1 Detailed Description . . . 2560
8.409.2 Constructor & Destructor Documentation . . . 2560
8.409.2.1 TransportUnicastSettings() [1/2] . . . 2560
8.409.2.2 TransportUnicastSettings() [2/2] . . . 2560
8.409.3 Member Function Documentation . . . 2561
8.409.3.1 transports() [1/2] . . . 2561
8.409.3.2 transports() [2/2] . . . 2561

8.409.3.3 receive_port() [1/2] . . . 2561
8.409.3.4 receive_port() [2/2] . . . 2562
8.410 rti::topic::trust::TrustAlgorithmRequirements Class Reference . . . 2562
8.410.1 Detailed Description . . . 2562
8.410.2 Constructor & Destructor Documentation . . . 2562
8.410.2.1 TrustAlgorithmRequirements() . . . 2562
8.410.3 Member Function Documentation . . . 2562
8.410.3.1 supported_mask() . . . 2563
8.410.3.2 required_mask() . . . 2563
8.411 dds::core::policy::TypeConsistencyEnforcement Class Reference . . . 2563
8.411.1 Detailed Description . . . 2564
8.411.2 Constructor & Destructor Documentation . . . 2565
8.411.2.1 TypeConsistencyEnforcement() [1/2] . . . 2565
8.411.2.2 TypeConsistencyEnforcement() [2/2] . . . 2565
8.411.3 Member Function Documentation . . . 2565
8.411.3.1 kind() [1/2] . . . 2566
8.411.3.2 kind() [2/2] . . . 2566
8.411.3.3 ignore_sequence_bounds() [1/2] . . . 2566
8.411.3.4 ignore_sequence_bounds() [2/2] . . . 2567
8.411.3.5 ignore_string_bounds() [1/2] . . . 2567
8.411.3.6 ignore_string_bounds() [2/2] . . . 2567
8.411.3.7 ignore_member_names() [1/2] . . . 2567
8.411.3.8 ignore_member_names() [2/2] . . . 2568
8.411.3.9 prevent_type_widening() [1/2] . . . 2568
8.411.3.10 prevent_type_widening() [2/2] . . . 2568
8.411.3.11 force_type_validation() [1/2] . . . 2568
8.411.3.12 force_type_validation() [2/2] . . . 2569
8.411.3.13 ignore_enum_literal_names() [1/2] . . . 2569
8.411.3.14 ignore_enum_literal_names() [2/2] . . . 2569
8.411.3.15 AllowTypeCoercion() . . . 2569
8.411.3.16 DisallowTypeCoercion() . . . 2570
8.411.3.17 AutoTypeCoercion() . . . 2570
8.412 dds::core::policy::TypeConsistencyEnforcementKind_def Struct Reference . . . 2570
8.412.1 Detailed Description . . . 2570
8.412.2 Member Enumeration Documentation . . . 2570
8.412.2.1 type . . . 2570
8.413 dds::core::xtypes::TypeKind_def Struct Reference . . . 2571
8.413.1 Detailed Description . . . 2571
8.413.2 Member Enumeration Documentation . . . 2572

8.413.2.1 type . . . 2572
8.414 rti::core::policy::TypeSupport Class Reference . . . 2574
8.414.1 Detailed Description . . . 2574
8.414.2 Usage . . . 2575
8.414.3 Constructor & Destructor Documentation . . . 2575
8.414.3.1 TypeSupport() . . . 2575
8.414.4 Member Function Documentation . . . 2575
8.414.4.1 plugin_data() [1/2] . . . 2575
8.414.4.2 plugin_data() [2/2] . . . 2576
8.414.4.3 cdr_padding_kind() [1/2] . . . 2576
8.414.4.4 cdr_padding_kind() [2/2] . . . 2576
8.415 dds::core::xtypes::UnidimensionalCollectionType Class Reference . . . 2576
8.415.1 Detailed Description . . . 2577
8.415.2 Member Function Documentation . . . 2577
8.415.2.1 bounds() . . . 2577
8.416 rti::flat::UnionBuilder< Discriminator > Class Template Reference . . . 2577
8.416.1 Detailed Description . . . 2578
8.417 dds::core::xtypes::UnionMember Class Reference . . . 2578
8.417.1 Detailed Description . . . 2579
8.417.2 Member Typedef Documentation . . . 2579
8.417.2.1 DiscriminatorType . . . 2579
8.417.2.2 LabelSeq . . . 2580
8.417.3 Constructor & Destructor Documentation . . . 2580
8.417.3.1 UnionMember() [1/4] . . . 2580
8.417.3.2 UnionMember() [2/4] . . . 2580
8.417.3.3 UnionMember() [3/4] . . . 2580
8.417.3.4 UnionMember() [4/4] . . . 2581
8.417.4 Member Function Documentation . . . 2581
8.417.4.1 name() [1/3] . . . 2581
8.417.4.2 name() [2/3] . . . 2581
8.417.4.3 type() . . . 2581
8.417.4.4 has_id() . . . 2581
8.417.4.5 get_id() . . . 2582
8.417.4.6 is_pointer() . . . 2582
8.417.4.7 label_count() . . . 2582
8.417.4.8 labels() [1/2] . . . 2582
8.417.4.9 labels() [2/2] . . . 2582
8.417.4.10 label() . . . 2583
8.417.4.11 name() [3/3] . . . 2583

8.417.4.12 id() . . . 2583
8.417.4.13 pointer() . . . 2583
8.417.5 Member Data Documentation . . . 2583
8.417.5.1 DEFAULT_LABEL . . . 2583
8.417.5.2 INVALID_ID . . . 2584
8.418 dds::core::xtypes::UnionType Class Reference . . . 2584
8.418.1 Detailed Description . . . 2585
8.418.2 Member Typedef Documentation . . . 2585
8.418.2.1 DiscriminatorType . . . 2585
8.418.3 Constructor & Destructor Documentation . . . 2585
8.418.3.1 UnionType() [1/4] . . . 2585
8.418.3.2 UnionType() [2/4] . . . 2586
8.418.3.3 UnionType() [3/4] . . . 2586
8.418.3.4 UnionType() [4/4] . . . 2587
8.418.4 Member Function Documentation . . . 2587
8.418.4.1 discriminator() . . . 2587
8.418.4.2 find_member_by_label() . . . 2587
8.418.4.3 find_member_by_id() . . . 2588
8.418.4.4 add_member() [1/2] . . . 2588
8.418.4.5 add_members() [1/3] . . . 2588
8.418.4.6 add_members() [2/3] . . . 2588
8.418.4.7 add_members() [3/3] . . . 2589
8.418.4.8 add_member() [2/2] . . . 2589
8.419 rti::core::UnregisterThreadOnExit Class Reference . . . 2589
8.419.1 Detailed Description . . . 2589
8.419.2 Constructor & Destructor Documentation . . . 2590
8.419.2.1 ∼UnregisterThreadOnExit() . . . 2590
8.420 dds::core::UnsupportedError Class Reference . . . 2590
8.420.1 Detailed Description . . . 2590
8.420.2 Member Function Documentation . . . 2590
8.420.2.1 what() . . . 2591
8.421 dds::core::policy::UserData Class Reference . . . 2591
8.421.1 Detailed Description . . . 2592
8.421.2 Usage . . . 2592
8.421.3 Constructor & Destructor Documentation . . . 2592
8.421.3.1 UserData() [1/3] . . . 2592
8.421.3.2 UserData() [2/3] . . . 2592
8.421.3.3 UserData() [3/3] . . . 2593
8.421.4 Member Function Documentation . . . 2593

8.421.4.1 value() [1/3] . . . 2593
8.421.4.2 value() [2/3] . . . 2594
8.421.4.3 value() [3/3] . . . 2594
8.421.4.4 begin() . . . 2594
8.421.4.5 end() . . . 2594
8.422 rti::sub::ValidLoanedSamples< T > Class Template Reference . . . 2595
8.422.1 Detailed Description . . . 2596
8.422.2 Member Typedef Documentation . . . 2596
8.422.2.1 iterator . . . 2596
8.422.2.2 const_iterator . . . 2596
8.422.2.3 value_type . . . 2597
8.422.3 Constructor & Destructor Documentation . . . 2597
8.422.3.1 ValidLoanedSamples() [1/2] . . . 2597
8.422.3.2 ValidLoanedSamples() [2/2] . . . 2597
8.422.4 Member Function Documentation . . . 2597
8.422.4.1 begin() [1/2] . . . 2598
8.422.4.2 begin() [2/2] . . . 2598
8.422.4.3 end() [1/2] . . . 2598
8.422.4.4 end() [2/2] . . . 2598
8.422.5 Friends And Related Function Documentation . . . 2598
8.422.5.1 begin() [1/2] . . . 2599
8.422.5.2 begin() [2/2] . . . 2599
8.422.5.3 end() [1/2] . . . 2599
8.422.5.4 end() [2/2] . . . 2600
8.422.5.5 swap() . . . 2600
8.423 rti::sub::ValidSampleIterator< T > Class Template Reference . . . 2600
8.423.1 Detailed Description . . . 2600
8.424 dds::core::Value< D > Class Template Reference . . . 2601
8.424.1 Detailed Description . . . 2601
8.424.2 Member Function Documentation . . . 2601
8.424.2.1 operator->() [1/2] . . . 2601
8.424.2.2 operator->() [2/2] . . . 2601
8.424.2.3 delegate() [1/2] . . . 2602
8.424.2.4 delegate() [2/2] . . . 2602
8.424.2.5 extensions() [1/2] . . . 2602
8.424.2.6 extensions() [2/2] . . . 2602
8.424.2.7 operator D&() . . . 2603
8.424.2.8 operator const D &() . . . 2603
8.425 dds::core::vector< T > Class Template Reference . . . 2603

8.425.1 Detailed Description . . . 2605
8.425.2 Constructor & Destructor Documentation . . . 2605
8.425.2.1 vector() [1/6] . . . 2605
8.425.2.2 vector() [2/6] . . . 2605
8.425.2.3 vector() [3/6] . . . 2606
8.425.2.4 vector() [4/6] . . . 2606
8.425.2.5 vector() [5/6] . . . 2606
8.425.2.6 vector() [6/6] . . . 2606
8.425.3 Member Function Documentation . . . 2606
8.425.3.1 operator std::vector< T >() . . . 2607
8.425.3.2 size() . . . 2607
8.425.3.3 capacity() . . . 2607
8.425.3.4 resize() [1/2] . . . 2607
8.425.3.5 resize() [2/2] . . . 2607
8.425.3.6 clear() . . . 2608
8.425.3.7 reserve() . . . 2608
8.425.3.8 at() [1/2] . . . 2608
8.425.3.9 at() [2/2] . . . 2608
8.425.3.10 operator[]() [1/2] . . . 2609
8.425.3.11 operator[]() [2/2] . . . 2609
8.425.3.12 operator=() [1/2] . . . 2609
8.425.3.13 operator=() [2/2] . . . 2609
8.425.3.14 operator==() . . . 2610
8.425.3.15 operator"!=() . . . 2610
8.425.3.16 begin() [1/2] . . . 2610
8.425.3.17 begin() [2/2] . . . 2610
8.425.3.18 end() [1/2] . . . 2610
8.425.3.19 end() [2/2] . . . 2611
8.425.4 Friends And Related Function Documentation . . . 2611
8.425.4.1 swap . . . 2611
8.425.4.2 operator<<() [1/2] . . . 2611
8.425.4.3 operator<<() [2/2] . . . 2611
8.426 rti::core::VendorId Class Reference . . . 2612
8.426.1 Detailed Description . . . 2612
8.426.2 Constructor & Destructor Documentation . . . 2612
8.426.2.1 VendorId() . . . 2612
8.426.3 Member Function Documentation . . . 2612
8.426.3.1 value() . . . 2612
8.426.3.2 unknown() . . . 2613

8.427 rti::config::Verbosity_def Struct Reference . . . 2613
8.427.1 Detailed Description . . . 2613
8.427.2 Member Enumeration Documentation . . . 2613
8.427.2.1 type . . . 2613
8.428 dds::sub::status::ViewState Class Reference . . . 2614
8.428.1 Detailed Description . . . 2615
8.428.2 Member Typedef Documentation . . . 2615
8.428.2.1 MaskType . . . 2615
8.428.3 Constructor & Destructor Documentation . . . 2615
8.428.3.1 ViewState() . . . 2615
8.428.4 Member Function Documentation . . . 2616
8.428.4.1 new_view() . . . 2616
8.428.4.2 not_new_view() . . . 2616
8.428.4.3 any() . . . 2616
8.428.5 Friends And Related Function Documentation . . . 2616
8.428.5.1 operator<<() . . . 2617
8.429 dds::core::cond::WaitSet Class Reference . . . 2617
8.429.1 Detailed Description . . . 2618
8.429.2 Usage . . . 2618
8.429.3 Trigger State of a dds::core::cond::StatusCondition . . . 2619
8.429.4 Trigger State of a dds::sub::cond::ReadCondition . . . 2620
8.429.5 Trigger State of a dds::core::cond::GuardCondition . . . 2620
8.429.6 Member Typedef Documentation . . . 2621
8.429.6.1 ConditionSeq . . . 2621
8.429.7 Constructor & Destructor Documentation . . . 2621
8.429.7.1 WaitSet() . . . 2621
8.429.8 Member Function Documentation . . . 2621
8.429.8.1 WaitSetImpl() . . . 2621
8.429.8.2 wait() [1/4] . . . 2621
8.429.8.3 wait() [2/4] . . . 2622
8.429.8.4 wait() [3/4] . . . 2622
8.429.8.5 wait() [4/4] . . . 2623
8.429.8.6 dispatch() [1/2] . . . 2623
8.429.8.7 dispatch() [2/2] . . . 2624
8.429.8.8 operator+=() . . . 2624
8.429.8.9 operator-=() . . . 2624
8.429.8.10 attach_condition() . . . 2625
8.429.8.11 detach_condition() . . . 2625
8.429.8.12 conditions() [1/2] . . . 2626

8.429.8.13 conditions() [2/2] . . . 2626
8.429.8.14 wait_one() . . . 2626
8.429.8.15 detach_all() . . . 2627
8.429.8.16 property() [1/2] . . . 2627
8.429.8.17 property() [2/2] . . . 2627
8.430 rti::core::cond::WaitSetProperty Class Reference . . . 2627
8.430.1 Detailed Description . . . 2628
8.430.2 Constructor & Destructor Documentation . . . 2628
8.430.2.1 WaitSetProperty() [1/2] . . . 2629
8.430.2.2 WaitSetProperty() [2/2] . . . 2629
8.430.3 Member Function Documentation . . . 2629
8.430.3.1 max_event_count() [1/2] . . . 2629
8.430.3.2 max_event_count() [2/2] . . . 2629
8.430.3.3 max_event_delay() [1/2] . . . 2630
8.430.3.4 max_event_delay() [2/2] . . . 2630
8.431 dds::core::WeakReference< T > Class Template Reference . . . 2630
8.431.1 Detailed Description . . . 2630
8.432 rti::core::policy::WireProtocol Class Reference . . . 2630
8.432.1 Detailed Description . . . 2632
8.432.2 Usage . . . 2632
8.432.3 Constructor & Destructor Documentation . . . 2634
8.432.3.1 WireProtocol() . . . 2634
8.432.4 Member Function Documentation . . . 2635
8.432.4.1 participant_id() [1/2] . . . 2635
8.432.4.2 participant_id() [2/2] . . . 2636
8.432.4.3 rtps_host_id() [1/2] . . . 2636
8.432.4.4 rtps_host_id() [2/2] . . . 2636
8.432.4.5 rtps_app_id() [1/2] . . . 2637
8.432.4.6 rtps_app_id() [2/2] . . . 2637
8.432.4.7 rtps_instance_id() [1/2] . . . 2638
8.432.4.8 rtps_instance_id() [2/2] . . . 2638
8.432.4.9 rtps_well_known_ports() [1/2] . . . 2638
8.432.4.10 rtps_well_known_ports() [2/2] . . . 2639
8.432.4.11 rtps_reserved_port_mask() [1/2] . . . 2639
8.432.4.12 rtps_reserved_port_mask() [2/2] . . . 2639
8.432.4.13 rtps_auto_id_kind() [1/2] . . . 2639
8.432.4.14 rtps_auto_id_kind() [2/2] . . . 2640
8.432.4.15 compute_crc() [1/2] . . . 2640
8.432.4.16 compute_crc() [2/2] . . . 2640

8.432.4.17 check_crc() [1/2] . . . 2640
8.432.4.18 check_crc() [2/2] . . . 2641
8.432.5 Member Data Documentation . . . 2641
8.432.5.1 RTPS_AUTO_ID . . . 2641
8.433 rti::core::policy::WireProtocolAutoKind_def Struct Reference . . . 2641
8.433.1 Detailed Description . . . 2642
8.433.2 Member Enumeration Documentation . . . 2642
8.433.2.1 type . . . 2642
8.434 rti::pub::WriteParams Class Reference . . . 2642
8.434.1 Detailed Description . . . 2644
8.434.2 Constructor & Destructor Documentation . . . 2644
8.434.2.1 WriteParams() . . . 2644
8.434.3 Member Function Documentation . . . 2644
8.434.3.1 reset() . . . 2644
8.434.3.2 replace_automatic_values() [1/2] . . . 2644
8.434.3.3 replace_automatic_values() [2/2] . . . 2645
8.434.3.4 identity() [1/2] . . . 2646
8.434.3.5 identity() [2/2] . . . 2646
8.434.3.6 related_sample_identity() [1/2] . . . 2646
8.434.3.7 related_sample_identity() [2/2] . . . 2647
8.434.3.8 source_timestamp() [1/2] . . . 2647
8.434.3.9 source_timestamp() [2/2] . . . 2647
8.434.3.10 cookie() [1/2] . . . 2647
8.434.3.11 cookie() [2/2] . . . 2648
8.434.3.12 handle() [1/2] . . . 2648
8.434.3.13 handle() [2/2] . . . 2648
8.434.3.14 priority() [1/2] . . . 2648
8.434.3.15 priority() [2/2] . . . 2649
8.434.3.16 flag() [1/2] . . . 2649
8.434.3.17 flag() [2/2] . . . 2650
8.434.3.18 source_guid() [1/2] . . . 2650
8.434.3.19 source_guid() [2/2] . . . 2650
8.434.3.20 related_source_guid() [1/2] . . . 2651
8.434.3.21 related_source_guid() [2/2] . . . 2651
8.434.3.22 related_reader_guid() [1/2] . . . 2651
8.434.3.23 related_reader_guid() [2/2] . . . 2652
8.435 rti::topic::WriterContentFilter< T, CompileData, WriterFilterData > Class Template Reference . . . 2652
8.435.1 Detailed Description . . . 2653
8.435.2 Member Function Documentation . . . 2654

8.435.2.1 writer_compile() . . . 2654
8.435.2.2 writer_evaluate() . . . 2655
8.435.2.3 writer_finalize() . . . 2655
8.435.2.4 writer_attach() . . . 2656
8.435.2.5 writer_detach() . . . 2656
8.435.2.6 writer_return_loan() . . . 2657
8.436 rti::topic::WriterContentFilterHelper< T, CompileData, WriterFilterData > Class Template Reference . 2657
8.436.1 Detailed Description . . . 2658
8.436.2 Member Function Documentation . . . 2658
8.436.2.1 writer_evaluate_helper() . . . 2659
8.436.2.2 add_cookie() . . . 2659
8.437 dds::core::policy::WriterDataLifecycle Class Reference . . . 2660
8.437.1 Detailed Description . . . 2660
8.437.2 Usage . . . 2661
8.437.3 Constructor & Destructor Documentation . . . 2661
8.437.3.1 WriterDataLifecycle() [1/2] . . . 2661
8.437.3.2 WriterDataLifecycle() [2/2] . . . 2661
8.437.4 Member Function Documentation . . . 2661
8.437.4.1 autodispose_unregistered_instances() [1/2] . . . 2662
8.437.4.2 autodispose_unregistered_instances() [2/2] . . . 2662
8.437.4.3 AutoDisposeUnregisteredInstances() . . . 2662
8.437.4.4 ManuallyDisposeUnregisteredInstances() . . . 2662
8.437.4.5 autopurge_unregistered_instances_delay() [1/2] . . . 2663
8.437.4.6 autopurge_unregistered_instances_delay() [2/2] . . . 2664
8.437.4.7 autopurge_disposed_instances_delay() [1/2] . . . 2664
8.437.4.8 autopurge_disposed_instances_delay() [2/2] . . . 2664
8.438 dds::core::xtypes::WStringType Class Reference . . . 2665
8.438.1 Detailed Description . . . 2665
8.438.2 Constructor & Destructor Documentation . . . 2665
8.438.2.1 WStringType() . . . 2665
8.439 rti::config::compliance::XTypesMask Class Reference . . . 2666
8.439.1 Detailed Description . . . 2667
8.439.2 Member Typedef Documentation . . . 2668
8.439.2.1 MaskType . . . 2668
8.439.3 Constructor & Destructor Documentation . . . 2668
8.439.3.1 XTypesMask() [1/3] . . . 2668
8.439.3.2 XTypesMask() [2/3] . . . 2668
8.439.3.3 XTypesMask() [3/3] . . . 2669
8.439.4 Member Function Documentation . . . 2669

8.439.4.1 dheader_in_non_primitive_collections() . . . 2669
8.439.4.2 enum_as_primitive_in_collections() . . . 2669
8.439.4.3 parameter_length_with_padding() . . . 2670
8.439.4.4 encapsulation_options_with_padding() . . . 2670
8.439.4.5 accept_unknown_enum_value() . . . 2671
8.439.4.6 accept_unknown_discriminator() . . . 2671
8.439.4.7 select_default_discriminator() . . . 2671
8.439.4.8 sentinel_in_empty_union() . . . 2672
8.439.4.9 initialize_discriminator_to_default() . . . 2672
8.439.4.10 java_keyhash_750() . . . 2673
8.439.4.11 python_int8_730() . . . 2673
8.439.4.12 must_understand_in_key_and_discriminator() . . . 2674
8.439.4.13 vendor() . . . 2674
8.439.4.14 default_mask() . . . 2674

9 Example Documentation **2675**

9.1 Foo.idl . . . 2675
9.2 Foo.hpp . . . 2676
9.3 Foo_publisher.cxx . . . 2681
9.4 Foo_subscriber.cxx . . . 2682
9.5 USER_QOS_PROFILES.xml . . . 2684

Index **2685**

Chapter 1

RTI Connext

Core Libraries and Utilities

Real-Time Innovations, Inc.

RTI Connext is network middleware for real-time distributed applications. It provides the communications services that programmers need to distribute time-critical data between embedded and/or enterprise devices or nodes. RTI Connext uses the publish-subscribe communications model to make data distribution efficient and robust.

The RTI Connext Application Programming Interface (API) is based on the OMG's Data Distribution Service (DDS) specification, version 1.4. The most recent publication of this specification can be found in the Catalog of OMG Specifications under "Platform Categories".

This documentation is for the **RTI Connext Modern C++ API**, based on the ISO/IEC C++ 2003 Language DDS PSM (DDS-PSM-Cxx) specification, version 1.0. The RTI Connext Traditional C++ API is also available.

1.1 Getting Started

- Learn about the **conventions** (p. 350) used in the API (such as type semantics, C++11 support, use of standard and extension APIs...) and the **headers and namespaces** (p. 355)
- Look at the **programming tutorials** (p. 362). These are a good starting point: **Publication Example** (p. 304) and **Subscription Example** (p. 305).
- Browse the API reference: **RTI Connext DDS API Reference** (p. 357) and **Additional RTI Connext Communication Patterns** (p. 361).
- Browse, download, and build these RTI Community code examples.

1.2 Available Documentation.

The documentation for this release is provided in two forms: the API Reference HTML documentation and PDF documents. If you are new to RTI Connext, the **Documentation Roadmap** (p. 349) will provide direction on how to learn about this product.

1.2.1 The documents for the Core Libraries and Utilities are:

- **What's New.** An overview of the new features in this release.
- **Release Notes.** System requirements, compatibility, what's fixed in this release, and known issues.
- **Platform Notes.** Specific details, such as compilation setting and libraries, related to building and using RTI Connext on the various supported platforms.
- **Getting Started Guide.** Core value and concepts behind the product, taking you step-by-step through the creation of a simple example application. Developers should read this document first.
- **Code Generator User's Manual.** Information about using rtiddsgen to generate code from data types.
- **User's Manual.** Introduction to RTI Connext, product tour and conceptual presentation of the functionality of RTI Connext.
- **QoS Reference Guide.** A compact summary of supported Quality of Service (QoS) policies.
- **XML-Based Application Creation Getting Started Guide.** Details on how to use XML-Based Application Creation.
- **Extensible Types Guide.** Additional information about extensible types.
- See more `documentation` on RTI Community.

1.2.2 The API Reference HTML documentation contains:

- **RTI Connext DDS API Reference** (p. 357) - The RTI Connext API reference.
- **Additional RTI Connext Communication Patterns** (p. 361) - RTI Connext APIs for additional communication patterns
- **Programming How-To's** (p. 362) - Describes and shows the common tasks done using the API.

The API Reference HTML documentation can be accessed through the tree view in the left frame of the web browser window. The bulk of the documentation is found under the entry labeled "Modules".

1.3 Feedback and Support for this Release.

We welcome any input on how to improve RTI Connext to suit your needs. If you have questions or comments about this release, please visit the RTI Customer Portal at `https://support.rti.com`.

The Customer Portal provides access to RTI software, documentation, and support. It also allows you to log support cases. Furthermore, the portal provides detailed solutions and a free public knowledge base. To access the software, documentation or log support cases, the RTI Customer Portal requires a username and password. You will receive this in the email confirming your purchase. If you do not have this email, please contact `license@rti.com`. Resetting your login password can be done directly at the RTI Customer Portal.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Documentation Roadmap . . . 349
Conventions . . . 350
Namespaces and headers . . . 355
RTI Connext DDS API Reference . . . 357
Domain Module . . . 44
DomainParticipants . . . 45
Built-in Topics . . . 46
Participant Built-in Topics . . . 441
Topic Built-in Topics . . . 442
Publication Built-in Topics . . . 442
Subscription Built-in Topics . . . 443
ServiceRequest Built-in Topic . . . 566
Topic Module . . . 47
Topics . . . 48
Zero Copy Transfer Over Shared Memory . . . 50
Built-in Types . . . 50
Built-in Topic's Trust Types . . . 54
FlatData Topic-Types . . . 419
FlatData Builders . . . 409
FlatData Samples . . . 412
FlatData Offsets . . . 414
DynamicType and DynamicData . . . 439
Topic traits and data-type support . . . 444
Topic-type serialization and deserialization . . . 568
Custom Content Filters . . . 571
Publication Module . . . 55
Publishers . . . 55
Data Writers . . . 56
Flow Controllers . . . 57
Multi-channel DataWriters . . . 73

Subscription Module . . . 61
Subscribers . . . 63
DataReaders . . . 63
Read Conditions . . . 65
Query Conditions . . . 65
Topic Queries . . . 67
Data Samples . . . 66
Sample Information . . . 71
Data State . . . 71
SampleProcessor . . . 566
Infrastructure Module . . . 72
Clock Selection . . . 43
Conditions and WaitSets . . . 426
AsyncWaitSet . . . 507
Time Support . . . 427
Exceptions . . . 427
Safe Enumeration . . . 429
Status Kinds . . . 429
Supporting Types and Constants . . . 433
Builtin Qos Profiles . . . 462
QoS . . . 508
All QoS Policies . . . 508
QoS Policy Traits . . . 428
ASYNCHRONOUS_PUBLISHER . . . 515
AVAILABILITY . . . 516
BATCH . . . 517
DATABASE . . . 517
DATA_READER_PROTOCOL . . . 517
DATA_READER_RESOURCE_LIMITS . . . 518
DATA_REPRESENTATION . . . 519
DATA_WRITER_PROTOCOL . . . 521
DATA_WRITER_RESOURCE_LIMITS . . . 522
DATA_WRITER_TRANSFER_MODE . . . 523
DATA_TAG . . . 523
DEADLINE . . . 523
DESTINATION_ORDER . . . 524
DISCOVERY . . . 525
NDDS_DISCOVERY_PEERS . . . 559
DISCOVERY_CONFIG . . . 526
DOMAIN_PARTICIPANT_RESOURCE_LIMITS . . . 528
DURABILITY . . . 529
DURABILITY_SERVICE . . . 531
ENTITY_FACTORY . . . 532
ENTITY_NAME . . . 532
EVENT . . . 533
HISTORY . . . 533
GROUP_DATA . . . 534
LATENCY_BUDGET . . . 534
LIFESPAN . . . 535
LIVELINESS . . . 535
LOCATORFILTER . . . 536
LOGGING . . . 536
MONITORING . . . 536
MULTICHANNEL . . . 537

OWNERSHIP . . . 538
OWNERSHIP_STRENGTH . . . 539
PARTITION . . . 539
PRESENTATION . . . 540
PROFILE . . . 541
PROPERTY . . . 541
PUBLISH_MODE . . . 541
READER_DATA_LIFECYCLE . . . 543
RECEIVER_POOL . . . 543
RELIABILITY . . . 544
RESOURCE_LIMITS . . . 546
SERVICE . . . 546
SYSTEM_RESOURCE_LIMITS . . . 547
TIME_BASED_FILTER . . . 547
TOPIC_DATA . . . 548
TOPIC_QUERY_DISPATCH . . . 548
TRANSPORT_BUILTIN . . . 548
TRANSPORT_MULTICAST . . . 549
TRANSPORT_MULTICAST_MAPPING . . . 550
TRANSPORT_PRIORITY . . . 550
TRANSPORT_SELECTION . . . 551
TRANSPORT_UNICAST . . . 551
TYPE_CONSISTENCY_ENFORCEMENT . . . 552
TYPESUPPORT . . . 553
USER_DATA . . . 554
WRITER_DATA_LIFECYCLE . . . 554
WIRE_PROTOCOL . . . 555
QoS by Category . . . 514
Basic QoS . . . 104
DDS Specification QoS . . . 112
DDS Extension QoS . . . 116
Functional Categories . . . 156
Application Acknowledgement . . . 158
Application-Defined Metadata . . . 159
Application Startup . . . 161
Batching . . . 161
Concurrency . . . 164
Content Filtering . . . 165
Data Availability . . . 167
Debuggability . . . 172
Discovery . . . 177
Discovery Isolation . . . 183
Durability . . . 184
Flow Control . . . 188
Fragmentation . . . 191
Instance Lifecycle . . . 193
Sending Large Data . . . 197
Liveliness . . . 199
Logging . . . 201
Middleware Memory Management . . . 204
QoS Profile Management . . . 220
Reliability QoS . . . 222
Sample Memory Management QoS . . . 230
Sample Ordering . . . 234

Security . . . 235
Throughput . . . 236
Transports . . . 237
Topic Queries . . . 239
Type System . . . 240
QoS Mutability . . . 242
Immutable QoS . . . 243
Mutable QoS . . . 280
Other QoS Attributes . . . 286
Applied Per-Instance . . . 287
Discovery Metadata . . . 288
Matching . . . 293
Deprecated QoS . . . 296
Transports . . . 75
Installing Transport Plugins . . . 80
Built-in Transport Plugins . . . 83
UDP Transport Plugin definitions . . . 378
Shared Memory Transport . . . 380
UDPv4 Transport . . . 386
Real-Time WAN Transport . . . 394
UDPv6 Transport . . . 399
Creating New Transport Plugins . . . 84
Transport Plugins Configuration . . . 365
Transport Address . . . 372
Queries and Filters Syntax . . . 84
Logging and Version . . . 90
Logging . . . 455
Activity Context . . . 449
Version . . . 459
General Utilities and Compliance Configuration . . . 90
Compliance Configuration . . . 451
Discovery Snapshot . . . 573
Network Capture . . . 578
Heap Monitoring . . . 589
Other Utilities . . . 595
Observability Framework . . . 91
Monitoring Library 2.0 . . . 598
Durability and Persistence . . . 99
System Properties . . . 301
Configuring QoS Profiles with XML . . . 302
Additional RTI Connext Communication Patterns . . . 361
Request-Reply Pattern . . . 91
Requester . . . 92
Replier . . . 93
Queuing Pattern . . . 93
QueueProducer . . . 95
QueueConsumer . . . 95
QueueRequester . . . 96
QueueReplier . . . 96
Remote Procedure Call . . . 97
Client-side API . . . 407
Server-side API . . . 408
Utilities . . . 98

Programming How-To's . . . 362
Publication Example . . . 304
Subscription Example . . . 305
Participant Use Cases . . . 306
Topic Use Cases . . . 308
Publisher Use Cases . . . 310
DataWriter Use Cases . . . 311
Subscriber Use Cases . . . 315
DataReader Use Cases . . . 316
Entity Use Cases . . . 325
Waitset Use Cases . . . 330
Filter Use Cases . . . 331
Creating Custom Content Filters . . . 333
XML Application Creation . . . 342
Request-Reply Examples . . . 344
RPC Tutorial . . . 421
Built-in Types Examples . . . 601
Exceptions . . . 601
Qos Use Cases . . . 602
Qos Provider Use Cases . . . 604
Working with IDL types . . . 606
DynamicType and DynamicData Use Cases . . . 609
Interface . . . 363

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

dds
The dds namespace. The dds namespace includes all of the standard types, classes, and functions 615
dds::all
<<***extension***>> *(p. 354)* A single namespace where all standard symbols are included 615
dds::core
The core namespace contains infrastructure types and functions 616
dds::core::cond
Contains the **Condition** (p. 948) classes 627
dds::core::policy
Contains the standard Qos policy classes 627
dds::core::status
Contains the Status and State classes 630
dds::core::xtypes
Contains the types and functions to support Extensible Types 631
dds::domain
The domain namespace contains types, classes, and functions related to DomainParticipants 634
dds::domain::qos
Contains **DomainParticipantQos** (p. 1384) 640
dds::pub
Contains the type and functions to support publishing topics 645
dds::pub::qos
Contains **PublisherQos** (p. 2007) and **DataWriterQos** (p. 1226) 653
dds::sub
Contains the types and functions to support subscribing to topics 658
dds::sub::cond
Contains Conditions specific to DataReaders 680
dds::sub::qos
Contains **DataReaderQos** (p. 1072) and **SubscriberQos** (p. 2423) 681
dds::sub::status
Contains **DataState** (p. 1118) 686

dds::topic
Contains topic related classes and functions, the built-in topics and topic traits used by IDL-generated types . . . 688
dds::topic::qos
Contains **TopicQos** (p. 2509) . . . 695
rti
<<***extension***>> *(p. 354)* The namespace that contains the extension types and functions to the DDS standard . . . 698
rti::all
<<***extension***>> *(p. 354)* A single namespace where all symbols are included . . . 699
rti::config
<<***extension***>> *(p. 354)* Logging configuration and version information . . . 699
rti::core
<<***extension***>> *(p. 354)* Extensions to **dds::core** (p. 616) . . . 701
rti::core::builtin_profiles
<<***extension***>> *(p. 354)* Contains the names of the built-in profiles . . . 711
rti::core::builtin_profiles::qos_lib
<<***extension***>> *(p. 354)* Contains the names of the built-in profiles in the regular (non-experimental) qos library . . . 712
rti::core::builtin_profiles::qos_lib_exp
<<***extension***>> *(p. 354)* <<***experimental***>> *(p. 355)* Contains the names of the built-in profiles in the experimental qos library . . . 714
rti::core::builtin_profiles::qos_snippet_lib
<<***extension***>> *(p. 354)* Contains the names of the built-in QoS Snippets . . . 716
rti::core::xtypes
<<***extension***>> *(p. 354)* Extensions to **dds::core::xtypes** (p. 631) . . . 718
rti::domain
<<***extension***>> *(p. 354)* Extensions to **dds::domain** (p. 634) . . . 727
rti::flat
<<***extension***>> *(p. 354)* Support for **FlatData Topic-Types** (p. 419) . . . 735
rti::pub
<<***extension***>> *(p. 354)* Extensions to **dds::pub** (p. 645) . . . 736
rti::sub
<<***extension***>> *(p. 354)* Extensions to **dds::sub** (p. 658) . . . 751
rti::topic
<<***extension***>> *(p. 354)* Extensions to **dds::topic** (p. 688) . . . 771
rti::util
<<***extension***>> *(p. 354)* Contains general-purpose utilities . . . 777
rti::util::activation
<<***extension***>> *(p. 354)* Contains the necessary functions related to Activation . . . 778
rti::util::discovery
<<***extension***>> *(p. 354)* Contains the functions to take discovery snapshot for DomainParticipant, DataWriter and DataReader entities . . . 778
rti::util::function_history
<<***extension***>> *(p. 354)* Contains the functions to enable Function History feature . . . 778
rti::util::heap_monitoring
<<***extension***>> *(p. 354)* Contains the functions to enable and use the heap-monitoring utility . . . 779
rti::util::network_capture
<<***extension***>> *(p. 354)* Contains the functions to enable and use the Network Capture utility . . . 780

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

rti::flat::AbstractBuilder . . . 783
rti::flat::AbstractListBuilder< detail::DHeaderGenerator > . . . 791
rti::flat::AbstractSequenceBuilder< detail::DHeaderGenerator > . . . 794
rti::flat::FinalSequenceBuilder< ElementOffset > . . . 1569
rti::flat::MutableSequenceBuilder< ElementBuilder > . . . 1757
rti::flat::MutableArrayBuilder< ElementBuilder, N > . . . 1753
rti::flat::AbstractListBuilder< detail::primitive_sequence_dheader_gen< T >::type > . . . 791
rti::flat::AbstractSequenceBuilder< detail::primitive_sequence_dheader_gen< T >::type > . . . 794
rti::flat::PrimitiveSequenceBuilder< T > . . . 1955
rti::flat::AbstractListBuilder< detail::primitive_sequence_dheader_gen< char >::type > . . . 791
rti::flat::AbstractSequenceBuilder< detail::primitive_sequence_dheader_gen< char >::type > . . . 794
rti::flat::PrimitiveSequenceBuilder< char > . . . 1955
rti::flat::StringBuilder . . . 2392
rti::flat::AbstractListBuilder< DHeaderGen > . . . 791
rti::flat::AbstractSequenceBuilder< DHeaderGen > . . . 794
rti::flat::AggregationBuilder . . . 800
rti::flat::UnionBuilder< int32_t > . . . 2577
MyFlatUnionBuilder . . . 1777
MyFlatMutableBuilder . . . 1763
rti::flat::UnionBuilder< Discriminator > . . . 2577
rti::pub::AcknowledgmentInfo . . . 795
rti::core::policy::AcknowledgmentKind_def . . . 797
rti::sub::AckResponseData . . . 798
rti::core::AllocationSettings . . . 803
dds::sub::AnyDataReader . . . 807
dds::sub::AnyDataReaderListener . . . 812
dds::sub::SubscriberListener . . . 2422
dds::domain::DomainParticipantListener . . . 1382
dds::domain::NoOpDomainParticipantListener . . . 1845

dds::sub::NoOpSubscriberListener . . . 1864
dds::domain::NoOpDomainParticipantListener . . . 1845
dds::pub::AnyDataWriter . . . 815
dds::pub::AnyDataWriterListener . . . 821
dds::pub::PublisherListener . . . 2006
dds::domain::DomainParticipantListener . . . 1382
dds::pub::NoOpPublisherListener . . . 1852
dds::domain::NoOpDomainParticipantListener . . . 1845
dds::topic::AnyTopic . . . 825
rti::core::policy::AsynchronousPublisher . . . 832
rti::core::cond::AsyncWaitSetListener . . . 856
rti::core::cond::NoOpAsyncWaitSetListener . . . 1836
rti::core::cond::AsyncWaitSetProperty . . . 858
rti::util::atomic_shared_ptr< T > . . . 862
rti::util::atomic_shared_ptr< detail::DRHolderBase > . . . 862
rti::util::atomic_shared_ptr< detail::DWHolderBase > . . . 862
rti::config::activity_context::AttributeKindMask . . . 863
rti::core::policy::Availability . . . 868
dds::core::basic_string< CharType, Allocator > . . . 875
rti::core::policy::Batch . . . 880
rti::core::bounded_sequence< T, MaxLength > . . . 887
dds::topic::BuiltinTopicKey . . . 905
rti::core::policy::BuiltinTopicReaderResourceLimits . . . 906
dds::core::BytesTopicType . . . 917
rti::core::policy::CdrPaddingKind_def . . . 920
rti::core::ChannelSettings . . . 921
dds::rpc::ClientParams . . . 928
dds::sub::CoherentAccess . . . 929
dds::pub::CoherentSet . . . 932
rti::core::CoherentSetInfo . . . 934
rti::core::CompressionIdMask . . . 940
rti::core::CompressionSettings . . . 943
dds::core::cond::Condition . . . 948
dds::core::cond::GuardCondition . . . 1594
dds::core::cond::StatusCondition . . . 2372
dds::sub::cond::ReadCondition . . . 2133
dds::sub::cond::QueryCondition . . . 2059
rti::queuing::ConsumerAvailabilityParams . . . 949
rti::topic::ContentFilter< T, CompileData > . . . 950
rti::topic::ContentFilter< T, no_compile_data_t > . . . 950
rti::topic::WriterContentFilter< T, no_compile_data_t, no_compile_data_t > . . . 2652
rti::topic::WriterContentFilterHelper< T, CompileData, WriterFilterData > . . . 2657
rti::topic::WriterContentFilter< T, CompileData, WriterFilterData > . . . 2652
rti::core::ContentFilterProperty . . . 960
rti::util::network_capture::ContentKindMask . . . 962
rti::core::Cookie . . . 965
rti::core::policy::Database . . . 969
rti::core::status::DataReaderCacheStatus . . . 1046
rti::core::policy::DataReaderInstanceRemovalKind_def . . . 1053
dds::sub::DataReaderListener< T > . . . 1055
dds::sub::NoOpDataReaderListener< T > . . . 1838
dds::sub::DataReaderListener< RequestType > . . . 1055

dds::sub::NoOpDataReaderListener< RequestType > . . . 1838
rti::core::policy::DataReaderProtocol . . . 1059
rti::core::status::DataReaderProtocolStatus . . . 1065
dds::sub::qos::DataReaderQos . . . 1072
rti::core::policy::DataReaderResourceLimits . . . 1081
rti::core::DataReaderResourceLimitsInstanceReplacementSettings . . . 1108
rti::sub::cond::DataReaderStatusConditionHandler< T > . . . 1111
dds::core::policy::DataRepresentation . . . 1113
dds::sub::status::DataState . . . 1118
rti::sub::status::DataStateEx . . . 1126
dds::core::policy::DataTag . . . 1132
rti::core::status::DataWriterCacheStatus . . . 1199
dds::pub::DataWriterListener< T > . . . 1202
dds::pub::NoOpDataWriterListener< T > . . . 1841
rti::core::policy::DataWriterProtocol . . . 1210
rti::core::status::DataWriterProtocolStatus . . . 1219
dds::pub::qos::DataWriterQos . . . 1226
rti::core::policy::DataWriterResourceLimits . . . 1235
rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_def . . . 1249
rti::core::DataWriterShmemRefTransferModeSettings . . . 1251
rti::core::policy::DataWriterTransferMode . . . 1253
dds::core::policy::Deadline . . . 1255
dds::core::policy::DestinationOrder . . . 1258
dds::core::policy::DestinationOrderKind_def . . . 1263
rti::core::policy::DestinationOrderScopeKind_def . . . 1264
rti::test::DisableMonitoringLibraryGuard . . . 1265
rti::core::policy::Discovery . . . 1265
rti::core::policy::DiscoveryConfig . . . 1271
rti::core::policy::DiscoveryConfigBuiltinChannelKindMask . . . 1319
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask . . . 1322
rti::domain::DomainParticipantConfigParams . . . 1371
dds::domain::qos::DomainParticipantFactoryQos . . . 1378
rti::core::status::DomainParticipantProtocolStatus . . . 1383
dds::domain::qos::DomainParticipantQos . . . 1384
rti::core::policy::DomainParticipantResourceLimits . . . 1391
dds::core::policy::Durability . . . 1433
dds::core::policy::DurabilityKind_def . . . 1441
dds::core::policy::DurabilityService . . . 1443
dds::core::Duration . . . 1448
rti::topic::dynamic_type< TopicType > . . . 1461
dds::core::xtypes::DynamicData . . . 1461
rti::core::xtypes::DynamicDataInfo . . . 1494
rti::core::xtypes::DynamicDataMemberInfo . . . 1495
rti::core::xtypes::DynamicDataProperty . . . 1497
rti::core::xtypes::DynamicDataTypeSerializationProperty . . . 1500
dds::core::xtypes::DynamicType . . . 1504
dds::core::xtypes::AbstractConstructedType< EnumMember > . . . 785
dds::core::xtypes::EnumType . . . 1535
dds::core::xtypes::AbstractConstructedType< Member > . . . 785
dds::core::xtypes::StructType . . . 2400
dds::core::xtypes::AbstractConstructedType< UnionMember > . . . 785
dds::core::xtypes::UnionType . . . 2584
dds::core::xtypes::AbstractConstructedType< MemberType > . . . 785

dds::core::xtypes::AliasType . . . 801
dds::core::xtypes::CollectionType . . . 939
dds::core::xtypes::ArrayType . . . 828
dds::core::xtypes::UnidimensionalCollectionType . . . 2576
dds::core::xtypes::SequenceType . . . 2344
dds::core::xtypes::StringType . . . 2399
dds::core::xtypes::WStringType . . . 2665
dds::core::xtypes::PrimitiveType . . . 1959
rti::core::xtypes::DynamicTypePrintFormatProperty . . . 1509
rti::core::EndpointGroup . . . 1515
rti::topic::trust::EndpointTrustAlgorithmInfo . . . 1516
rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo . . . 1517
rti::topic::trust::EndpointTrustProtectionInfo . . . 1519
dds::core::Entity . . . 1520
dds::pub::DataWriter< dds::core::xtypes::DynamicData > . . . 1138
dds::pub::DataWriter< ReplyType > . . . 1138
dds::pub::DataWriter< RequestType > . . . 1138
dds::sub::DataReader< RequestType > . . . 976
dds::sub::DataReader< ReplyType > . . . 976
dds::domain::DomainParticipant . . . 1326
dds::pub::DataWriter< T > . . . 1138
dds::pub::Publisher . . . 1993
dds::sub::DataReader< T > . . . 976
dds::sub::Subscriber . . . 2410
dds::topic::TopicDescription< T > . . . 2503
dds::topic::ContentFilteredTopic< T > . . . 954
dds::topic::Topic< T > . . . 2474
dds::core::policy::EntityFactory . . . 1527
rti::core::policy::EntityName . . . 1530
dds::core::xtypes::EnumMember . . . 1533
rti::test::EnvVarToken . . . 1539
rti::core::policy::Event . . . 1541
rti::core::status::EventCount< IntegerType > . . . 1544
dds::core::Exception . . . 1546
dds::core::AlreadyClosedError . . . 806
dds::core::Error . . . 1539
dds::core::IllegalOperationError . . . 1608
dds::core::ImmutablePolicyError . . . 1609
dds::core::InconsistentPolicyError . . . 1610
dds::core::InvalidArgumentError . . . 1619
dds::core::InvalidDowncastError . . . 1620
dds::core::NotAllowedBySecurityError . . . 1868
dds::core::NotEnabledError . . . 1869
dds::core::NullReferenceError . . . 1870
dds::core::OutOfResourcesError . . . 1898
dds::core::PreconditionNotMetError . . . 1941
dds::core::TimeoutError . . . 2473
dds::core::UnsupportedError . . . 2590
dds::rpc::RemoteInvalidArgumentError . . . 2162
dds::rpc::RemoteOutOfResourcesError . . . 2163
dds::rpc::RemoteUnknownExceptionError . . . 2165
dds::rpc::RemoteUnknownOperationError . . . 2166
dds::rpc::RemoteUnsupportedOperationError . . . 2167

rti::queuing::NoMatchingQueueException . . . 1835
rti::topic::ExpressionProperty . . . 1547
rti::topic::extensibility< TopicType > . . . 1550
dds::core::xtypes::ExtensibilityKind_def . . . 1550
dds::core::external< T > . . . 1551
dds::topic::Filter . . . 1558
rti::topic::FilterSampleInfo . . . 1562
rti::flat::flat_type_traits< T > . . . 1571
rti::flat::flat_type_traits< PrimitiveSequenceOffset< char > > . . . 1571
rti::pub::FlowControllerProperty . . . 1577
rti::pub::FlowControllerSchedulingPolicy_def . . . 1581
rti::pub::FlowControllerTokenBucketProperty . . . 1583
Foo . . . 1588
dds::sub::GenerationCount . . . 1589
dds::core::policy::GroupData . . . 1591
rti::core::Guid . . . 1596
rti::util::heap_monitoring::HeapMonitoringParams . . . 1599
dds::core::policy::History . . . 1602
dds::core::policy::HistoryKind_def . . . 1606
rti::core::policy::IgnoredEntityReplacementKind_def . . . 1607
dds::core::status::InconsistentTopicStatus . . . 1611
dds::core::InstanceHandle . . . 1612
dds::sub::status::InstanceState . . . 1615
rti::core::status::InvalidLocalAdvanceNoticeStatus . . . 1621
dds::topic::is_topic_type< T > . . . 1622
rti::request::IsReplyRelatedPredicate< T > . . . 1622
rti::sub::IsValidData< T > . . . 1624
dds::core::KeyedBytesTopicType . . . 1626
dds::core::KeyedStringTopicType . . . 1629
dds::core::policy::LatencyBudget . . . 1631
rti::config::LibraryVersion . . . 1634
dds::core::policy::Lifespan . . . 1635
Listener . . . 1637
dds::core::policy::Liveliness . . . 1645
dds::core::status::LivelinessChangedStatus . . . 1652
dds::core::policy::LivelinessKind_def . . . 1654
dds::core::status::LivelinessLostStatus . . . 1655
rti::core::xtypes::LoanedDynamicData . . . 1656
rti::sub::LoanedSample< T > . . . 1659
dds::sub::LoanedSamples< T > . . . 1663
rti::core::Locator . . . 1673
rti::core::policy::LocatorFilter . . . 1676
rti::core::LocatorFilterElement . . . 1678
rti::core::LocatorKind_def . . . 1681
rti::config::LogCategory_def . . . 1682
rti::config::Logger . . . 1683
rti::config::LogLevel_def . . . 1692
rti::config::LogMessage . . . 1693
rti::core::LongDouble . . . 1696
dds::sub::DataReader< T >::ManipulatorSelector . . . 1697
rti::sub::ManipulatorSelector< T > . . . 1700
dds::core::xtypes::Member . . . 1700
rti::core::policy::Monitoring . . . 1706
rti::core::MonitoringDedicatedParticipantSettings . . . 1712

rti::core::MonitoringDistributionSettings . . . 1716
rti::core::MonitoringEventDistributionSettings . . . 1722
rti::core::MonitoringLoggingDistributionSettings . . . 1727
rti::core::MonitoringLoggingForwardingSettings . . . 1733
rti::core::MonitoringMetricSelection . . . 1736
rti::core::MonitoringPeriodicDistributionSettings . . . 1740
rti::core::MonitoringTelemetryData . . . 1744
rti::core::policy::MulticastMapping . . . 1747
rti::core::policy::MultiChannel . . . 1749
NDDS_Transport_Address_t . . . 1784
NDDS_Transport_Interface_t . . . 1785
NDDS_Transport_Property_t . . . 1786
NDDS_Transport_Shmem_Property_t . . . 1794
NDDS_Transport_UDP_WAN_CommPortsMappingInfo . . . 1797
NDDS_Transport_UDPv4_Property_t . . . 1798
NDDS_Transport_UDPv4_WAN_Property_t . . . 1809
NDDS_Transport_UDPv6_Property_t . . . 1819
NDDS_Transport_UUID . . . 1829
rti::util::network_capture::NetworkCaptureParams . . . 1829
rti::topic::no_compile_data_t . . . 1835
rti::core::xtypes::octet_tag_t . . . 1871
dds::core::status::OfferedDeadlineMissedStatus . . . 1871
dds::core::status::OfferedIncompatibleQosStatus . . . 1873
rti::flat::OffsetBase . . . 1875
  rti::flat::AbstractPrimitiveList< char > . . . 792
    rti::flat::PrimitiveSequenceOffset< char > . . . 1958
      rti::flat::StringOffset . . . 2394
  rti::flat::FinalOffset< MyFlatFinalOffset > . . . 1568
    MyFlatFinalOffset . . . 1759
  rti::flat::AbstractAlignedList< ElementOffset > . . . 781
    rti::flat::FinalAlignedArrayOffset< ElementOffset, N > . . . 1565
    rti::flat::MutableArrayOffset< ElementOffset, N > . . . 1755
    rti::flat::SequenceOffset< ElementOffset > . . . 2343
  rti::flat::AbstractPrimitiveList< T > . . . 792
    rti::flat::PrimitiveArrayOffset< T, N > . . . 1951
    rti::flat::PrimitiveSequenceOffset< T > . . . 1958
  rti::flat::FinalArrayOffset< ElementOffset, N > . . . 1567
  rti::flat::FinalOffset< T > . . . 1568
  rti::flat::MutableOffset . . . 1756
    MyFlatMutableOffset . . . 1770
    MyFlatUnionOffset . . . 1780
  rti::flat::PrimitiveConstOffset< T > . . . 1952
    rti::flat::PrimitiveOffset< T > . . . 1953
dds::core::optional< T > . . . 1879
dds::core::optional< dds::core::xtypes::DynamicType > . . . 1879
dds::core::optional< dds::pub::qos::DataWriterQos > . . . 1879
dds::core::optional< dds::sub::qos::DataReaderQos > . . . 1879
dds::core::optional< std::string > . . . 1879
rti::core::optional_value< T > . . . 1892
OsapiFile . . . 1898
dds::core::policy::Ownership . . . 1899
dds::core::policy::OwnershipKind_def . . . 1905
dds::core::policy::OwnershipStrength . . . 1906

dds::topic::ParticipantBuiltinTopicData . . . 1908
rti::test::ParticipantFactoryGuard . . . 1916
rti::topic::trust::ParticipantTrustAlgorithmInfo . . . 1916
rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo . . . 1917
rti::topic::trust::ParticipantTrustKeyEstablishmentAlgorithmInfo . . . 1919
rti::topic::trust::ParticipantTrustProtectionInfo . . . 1920
rti::topic::trust::ParticipantTrustSignatureAlgorithmInfo . . . 1922
dds::core::policy::Partition . . . 1923
rti::core::PersistentStorageSettings . . . 1927
rti::core::pointer< T > . . . 1938
dds::core::policy::policy_id< Policy > . . . 1940
dds::core::policy::policy_name< Policy > . . . 1941
dds::core::policy::Presentation . . . 1942
dds::core::policy::PresentationAccessScopeKind_def . . . 1950
rti::config::PrintFormat_def . . . 1960
rti::topic::PrintFormatKind_def . . . 1961
rti::topic::PrintFormatProperty . . . 1962
rti::core::ProductVersion . . . 1967
rti::core::policy::Property . . . 1969
rti::core::ProtocolVersion . . . 1975
dds::topic::PublicationBuiltinTopicData . . . 1977
dds::core::status::PublicationMatchedStatus . . . 1990
dds::pub::qos::PublisherQos . . . 2007
rti::core::policy::PublishMode . . . 2013
rti::core::policy::PublishModeKind_def . . . 2018
rti::core::qos_print_all_t . . . 2019
dds::core::policy::QosPolicyCount . . . 2020
rti::core::QosPrintFormat . . . 2021
dds::core::QosProvider . . . 2025
rti::core::QosProviderParams . . . 2049
dds::sub::Query . . . 2053
rti::queuing::QueueConsumerListener< T > . . . 2075
rti::queuing::NoOpQueueConsumerListener< T > . . . 1856
rti::queuing::QueueConsumerListener< TRep > . . . 2075
rti::queuing::NoOpQueueConsumerListener< TRep > . . . 1856
rti::queuing::QueueConsumerListener< TReq > . . . 2075
rti::queuing::NoOpQueueConsumerListener< TReq > . . . 1856
rti::queuing::QueueEntityParams< ActualEntity > . . . 2079
rti::queuing::QueueEntityParams< QueueConsumerParams > . . . 2079
rti::queuing::QueueConsumerParams . . . 2077
rti::queuing::QueueEntityParams< QueueProducerParams > . . . 2079
rti::queuing::QueueProducerParams . . . 2092
rti::queuing::QueueEntityParams< QueueReplierParams > . . . 2079
rti::queuing::QueueReplierParams . . . 2109
rti::queuing::QueueEntityParams< QueueRequesterParams > . . . 2079
rti::queuing::QueueRequesterParams . . . 2128
rti::queuing::QueueProducerListener< T > . . . 2090
rti::queuing::NoOpQueueProducerListener< T > . . . 1858
rti::queuing::QueueProducerListener< TRep > . . . 2090
rti::queuing::NoOpQueueProducerListener< TRep > . . . 1858
rti::queuing::QueueProducerListener< TReq > . . . 2090

rti::queuing::NoOpQueueProducerListener< TReq > 1858
rti::queuing::QueueReplierListener< TReq, TRep > 2107
rti::queuing::NoOpQueueReplierListener< TReq, TRep > 1859
rti::queuing::QueueRequesterListener< TReq, TRep > 2126
rti::queuing::NoOpQueueRequesterListener< TReq, TRep > 1861
dds::sub::Rank 2130
dds::core::policy::ReaderDataLifecycle 2137
dds::sub::ReadModeDummyType 2144
rti::core::policy::ReceiverPool 2144
dds::core::Reference< DELEGATE > 2149
dds::core::Reference< AsyncWaitSetCompletionTokenImpl > 2149
rti::core::cond::AsyncWaitSetCompletionToken 853
dds::core::Reference< AsyncWaitSetImpl > 2149
rti::core::cond::AsyncWaitSet 840
dds::core::Reference< ContentFilterBase > 2149
rti::topic::CustomFilter< ContentFilterBase > 967
dds::core::Reference< detail::ClientEndpointImpl< Request, Reply > > 2149
dds::rpc::ClientEndpoint< Request, Reply > 925
rpc_example::RobotControlClient 2217
dds::core::Reference< detail::QueueConsumerImpl< T > > 2149
rti::queuing::QueueConsumer< T > 2063
dds::core::Reference< detail::QueueConsumerImpl< TRep > > 2149
rti::queuing::QueueConsumer< TRep > 2063
dds::core::Reference< detail::QueueConsumerImpl< TReq > > 2149
rti::queuing::QueueConsumer< TReq > 2063
dds::core::Reference< detail::QueueProducerImpl< T > > 2149
rti::queuing::QueueProducer< T > 2080
dds::core::Reference< detail::QueueProducerImpl< TRep > > 2149
rti::queuing::QueueProducer< TRep > 2080
dds::core::Reference< detail::QueueProducerImpl< TReq > > 2149
rti::queuing::QueueProducer< TReq > 2080
dds::core::Reference< detail::QueueReplierImpl< TReq, TRep > > 2149
rti::queuing::QueueReplier< TReq, TRep > 2094
dds::core::Reference< detail::QueueRequesterImpl< TReq, TRep > > 2149
rti::queuing::QueueRequester< TReq, TRep > 2111
dds::core::Reference< detail::ReplierImpl< RequestType, ReplyType > > 2149
rti::request::Replier< RequestType, ReplyType > 2168
rti::request::SimpleReplier< RequestType, ReplyType > 2368
dds::core::Reference< detail::RequesterImpl< Request, Reply > > 2149
rti::request::Requester< Request, Reply > 2187
dds::core::Reference< detail::RequesterImpl< RequestType, ReplyType > > 2149
rti::request::Requester< RequestType, ReplyType > 2187
dds::core::Reference< detail::ServiceEndpointImpl< Dispatcher > > 2149
dds::rpc::ServiceEndpoint< Dispatcher > 2354
dds::core::Reference< DynamicDataProxyTypeSupportImpl > 2149
dds::core::Reference< FilterHolderImpl > 2149
dds::core::Reference< FlowControllerImpl > 2149
rti::pub::FlowController 1572

dds::core::Reference< SampleProcessorImpl > . . . 2149
rti::sub::SampleProcessor . . . 2305
dds::core::Reference< T > . . . 2149
rti::topic::CustomFilter< T > . . . 967
dds::core::Reference< TopicQueryImpl > . . . 2149
rti::sub::TopicQuery . . . 2516
dds::core::Reference< typename Entity::Listener > . . . 2149
rti::core::ListenerBinder< Entity, Listener > . . . 1641
dds::core::Reference< UserProxyTypeSupportImpl > . . . 2149
dds::core::policy::Reliability . . . 2149
dds::core::policy::ReliabilityKind_def . . . 2156
rti::core::status::ReliableReaderActivityChangedStatus . . . 2157
rti::core::status::ReliableWriterCacheChangedStatus . . . 2159
rti::core::policy::RemoteParticipantPurgeKind_def . . . 2164
rti::request::ReplierListener< RequestType, ReplyType > . . . 2179
rti::request::ReplierParams . . . 2180
dds::core::status::RequestedDeadlineMissedStatus . . . 2184
dds::core::status::RequestedIncompatibleQosStatus . . . 2185
rti::request::RequesterParams . . . 2199
dds::core::policy::ResourceLimits . . . 2203
rti::core::Result< T > . . . 2209
rpc_example::RobotControl . . . 2213
rpc_example::RobotControlClient . . . 2217
rpc_example::RobotControlAsync . . . 2215
rpc_example::RobotControlClient . . . 2217
rti::config::compliance::RpcMask . . . 2218
rti::core::RtpsReliableReaderProtocol . . . 2221
rti::core::policy::RtpsReliableWriterProtocol . . . 2229
rti::core::policy::RtpsReservedPortKindMask . . . 2255
rti::core::RtpsWellKnownPorts . . . 2258
dds::core::safe_enum< def, inner > . . . 2265
dds::core::safe_enum< LogLevel_def > . . . 2265
dds::core::safe_enum< Verbosity_def > . . . 2265
dds::sub::Sample< T > . . . 2270
rti::flat::Sample< OffsetType > . . . 2274
rti::core::SampleFlag . . . 2278
rti::core::SampleIdentity . . . 2282
dds::sub::SampleInfo . . . 2285
rti::sub::SampleIterator< T > . . . 2296
rti::core::status::SampleLostState . . . 2296
dds::core::status::SampleLostStatus . . . 2303
dds::core::status::SampleRejectedState . . . 2310
dds::core::status::SampleRejectedStatus . . . 2315
dds::sub::status::SampleState . . . 2316
rti::config::ScopedLoggerVerbosity . . . 2319
dds::sub::DataReader< T >::Selector . . . 2320
rti::flat::SequenceIterator< E, OffsetKind > . . . 2328
rti::core::SequenceNumber . . . 2335
dds::rpc::Server . . . 2346
dds::rpc::ServerParams . . . 2348
rti::core::policy::Service . . . 2350
rti::core::policy::ServiceKind_def . . . 2357

rti::topic::ServiceRequest . . . 2358
rti::core::status::ServiceRequestAcceptedStatus . . . 2360
rti::core::ServiceRequestId_def . . . 2361
dds::sub::SharedSamples< T, DELEGATE > . . . 2362
rti::sub::SharedSamples< T > . . . 2367
rti::util::heap_monitoring::SnapshotContentFormat_def . . . 2370
rti::util::heap_monitoring::SnapshotOutputFormat_def . . . 2371
dds::core::status::StatusMask . . . 2375
rti::util::StreamFlagSaver . . . 2390
rti::sub::status::StreamKind . . . 2390
dds::core::StringTopicType . . . 2395
dds::sub::qos::SubscriberQos . . . 2423
dds::topic::SubscriptionBuiltinTopicData . . . 2428
dds::core::status::SubscriptionMatchedStatus . . . 2439
dds::pub::SuspendedPublication . . . 2441
rti::core::policy::SystemResourceLimits . . . 2443
rti::core::ThreadSettings . . . 2449
rti::core::ThreadSettingsCpuRotationKind_def . . . 2453
rti::core::ThreadSettingsKindMask . . . 2454
dds::core::Time . . . 2457
dds::core::policy::TimeBasedFilter . . . 2469
rti::topic::topic_type_disabled_copy< TopicType > . . . 2492
rti::topic::topic_type_has_external_members< TopicType > . . . 2492
dds::topic::topic_type_name< T > . . . 2493
dds::topic::topic_type_support< T > . . . 2493
dds::topic::TopicBuiltinTopicData . . . 2493
dds::core::policy::TopicData . . . 2500
dds::topic::TopicInstance< T > . . . 2505
dds::topic::TopicListener< T > . . . 2508
dds::topic::NoOpTopicListener< T > . . . 1867
dds::topic::qos::TopicQos . . . 2509
rti::sub::TopicQueryData . . . 2520
rti::core::policy::TopicQueryDispatch . . . 2522
rti::sub::TopicQuerySelection . . . 2526
rti::sub::TopicQuerySelectionKind_def . . . 2528
rti::util::network_capture::TrafficKindMask . . . 2529
TransportAllocationSettings_t . . . 2533
rti::core::policy::TransportBuiltin . . . 2533
rti::core::policy::TransportBuiltinMask . . . 2537
rti::core::TransportClassId_def . . . 2540
rti::core::TransportInfo . . . 2541
rti::core::policy::TransportMulticast . . . 2543
rti::core::policy::TransportMulticastKind_def . . . 2547
rti::core::policy::TransportMulticastMapping . . . 2547
rti::core::TransportMulticastSettings . . . 2549
dds::core::policy::TransportPriority . . . 2552
rti::core::policy::TransportSelection . . . 2554
rti::core::policy::TransportUnicast . . . 2556
rti::core::TransportUnicastSettings . . . 2559
rti::topic::trust::TrustAlgorithmRequirements . . . 2562
dds::core::policy::TypeConsistencyEnforcement . . . 2563
dds::core::policy::TypeConsistencyEnforcementKind_def . . . 2570
dds::core::xtypes::TypeKind_def . . . 2571
rti::core::policy::TypeSupport . . . 2574

dds::core::xtypes::UnionMember 2578
rti::core::UnregisterThreadOnExit 2589
dds::core::policy::UserData 2591
rti::sub::ValidLoanedSamples< T > 2595
rti::sub::ValidSampleIterator< T > 2600
dds::core::Value< D > 2601
dds::core::Value< DELEGATE > 2601
dds::core::TEntityQos< DELEGATE > 2446
dds::core::vector< T > 2603
dds::core::vector< uint8_t > 2603
rti::core::VendorId 2612
rti::config::Verbosity_def 2613
dds::sub::status::ViewState 2614
dds::core::cond::WaitSet 2617
rti::core::cond::WaitSetProperty 2627
dds::core::WeakReference< T > 2630
rti::core::policy::WireProtocol 2630
rti::core::policy::WireProtocolAutoKind_def 2641
rti::pub::WriteParams 2642
dds::core::policy::WriterDataLifecycle 2660
rti::config::compliance::XTypesMask 2666

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

rti::flat::AbstractAlignedList< ElementOffset >
Base class of Offsets to sequences and arrays of non-primitive members . . . 781
rti::flat::AbstractBuilder
Base class of all **Builders** (p. 409) . . . 783
dds::core::xtypes::AbstractConstructedType< MemberType >
The base class of types that have members and an extensibility kind . . . 785
rti::flat::AbstractListBuilder< DHeaderGen >
Base class of all array and sequence builders . . . 791
rti::flat::AbstractPrimitiveList< T >
Base class for Offsets to sequences and arrays of primitive types . . . 792
rti::flat::AbstractSequenceBuilder< DHeaderGen >
Base class of Builders for sequence members . . . 794
rti::pub::AcknowledgmentInfo
<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Information about an application-acknowledged sample . . . 795
rti::core::policy::AcknowledgmentKind_def
<<***extension***>> *(p. 354)* The enumeration for Reliability acknowledgment kinds . . . 797
rti::sub::AckResponseData
<<***extension***>> *(p. 354)* Data payload associated to an application-level acknowledgment . . . 798
rti::flat::AggregationBuilder
Base class of struct and union builders . . . 800
dds::core::xtypes::AliasType
<<***value-type***>> *(p. 350)* Represents an IDL `typedef` . . . 801
rti::core::AllocationSettings
<<***extension***>> *(p. 354)* Resource allocation settings . . . 803
dds::core::AlreadyClosedError
Indicates that an object has been closed . . . 806
dds::sub::AnyDataReader
<<***reference-type***>> *(p. 351)* This class provides an non-template holder for representing a **DataReader** (p. 976) of any type . . . 807
dds::sub::AnyDataReaderListener
The listener to notify status changes for a **dds::sub::DataReader** (p. 976) of a generic type . . . 812

dds::pub::AnyDataWriter
<<***reference-type***>> *(p. 351)* This class provides an non-template holder for representing a **DataWriter** (p. 1138) of any type . . . 815
dds::pub::AnyDataWriterListener
The listener to notify status changes for a **dds::pub::DataWriter** (p. 1138) of a generic type . . . 821
dds::topic::AnyTopic
<<***reference-type***>> *(p. 351)* This class provides an non-template holder for representing a **Topic** (p. 2474) of any type . . . 825
dds::core::xtypes::ArrayType
<<***value-type***>> *(p. 350)* Represents an IDL array type . . . 828
rti::core::policy::AsynchronousPublisher
<<***extension***>> *(p. 116)* Configures the mechanism to publish data using a separate thread . . . 832
rti::core::cond::AsyncWaitSet
<<***extension***>> *(p. 354)* Dispatches **dds::core::cond::Condition** (p. 948) objects using separate threads of execution. Unlike a normal **dds::core::cond::WaitSet** (p. 2617), which operates on the current thread, an **AsyncWaitSet** (p. 840) uses a thread pool . . . 840
rti::core::cond::AsyncWaitSetCompletionToken
Implementation of the completion token role element of the asynchronous completion token pattern that is part of the **AsyncWaitSet** (p. 840) behavior . . . 853
rti::core::cond::AsyncWaitSetListener
Listener (p. 1637) for receiving event notifications related to the thread pool of the **AsyncWaitSet** (p. 840) . . . 856
rti::core::cond::AsyncWaitSetProperty
Specifies the **rti::core::cond::AsyncWaitSet** (p. 840) behavior . . . 858
rti::util::atomic_shared_ptr< T >
Utility class to provide atomic operations on std::shared_ptr . . . 862
rti::config::activity_context::AttributeKindMask
The attributes used in the string representation of the Activity Context can be configured through this mask . . . 863
rti::core::policy::Availability
<<***extension***>> *(p. 116)* Configures data availability in the context of Collaborative DataWriters and Required Subscriptions . . . 868
dds::core::basic_string< CharType, Allocator >
<<***value-type***>> *(p. 350)* A string convertible to std::string and with similar functionality . . . 875
rti::core::policy::Batch
<<***extension***>> *(p. 116)* Allows a **dds::pub::DataWriter** (p. 1138) to batch multiple samples into a single network packet to increase throughput . . . 880
rti::core::bounded_sequence< T, MaxLength >
<<***value-type***>> *(p. 350)* A bounded sequence of elements . . . 887
dds::topic::BuiltinTopicKey
The key of the built-in topics . . . 905
rti::core::policy::BuiltinTopicReaderResourceLimits
<<***extension***>> *(p. 116)* Configures several resource management aspects of the built-in topic DataReaders . . . 906
dds::core::BytesTopicType
Built-in type consisting of a variable-length array of opaque bytes . . . 917
rti::core::policy::CdrPaddingKind_def
<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) CdrPaddingKind . 920
rti::core::ChannelSettings
<<***extension***>> *(p. 354)* Configures the properties of a channel in **rti::core::policy::Multi↩ Channel** (p. 1749) . . . 921
dds::rpc::ClientEndpoint< Request, Reply >
<<***reference-type***>> *(p. 351)* Manages the DDS entities required to make remote function calls . 925

dds::rpc::ClientParams
<<***value-type***>> *(p. 350)* The parameters used to configure a **ClientEndpoint** (p. 925) 928
dds::sub::CoherentAccess
<<***value-type***>> *(p. 350)* Controls whether RTI Connext will preserve the groupings of changes made by the publishing application by means of begin_coherent_changes and end_coherent_↩ changes 929
dds::pub::CoherentSet
<<***value-type***>> *(p. 350)* A publishing application can request that a set of DDS data-sample changes be propagated in such a way that they are interpreted at the receivers' side as a cohesive set of modifications 932
rti::core::CoherentSetInfo
<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* A CoherentSampleInfo provides information about the coherent set associated with a sample 934
dds::core::xtypes::CollectionType
<<***value-type***>> *(p. 350)* The base class of all collection types 939
rti::core::CompressionIdMask
<<***extension***>> *(p. 354)* Mask that specifies which built-in compression method to used 940
rti::core::CompressionSettings
<<***extension***>> *(p. 354)* Compression Settings 943
dds::core::cond::Condition
<<***reference-type***>> *(p. 351)* Abstract base class of all the conditions 948
rti::queuing::ConsumerAvailabilityParams
Definition of the availability feedback information that can be provided by consumers to Queuing Service 949
rti::topic::ContentFilter< T, CompileData >
<<***extension***>> *(p. 354)* A class to inherit from when implementing a custom content filter 950
dds::topic::ContentFilteredTopic< T >
<<***reference-type***>> *(p. 351)* Specialization of **TopicDescription** (p. 2503) that allows for content-based subscriptions 954
rti::core::ContentFilterProperty
<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Provides all the required information to enable content filtering 960
rti::util::network_capture::ContentKindMask
<<***extension***>> *(p. 354)* Mask indicating the types of contents to remove from RTPS frames before saving them to the capture file 962
rti::core::Cookie
Unique identifier for a written data sample in the form of a sequence of bytes 965
rti::topic::CustomFilter< T >
<<***extension***>> *(p. 354)* <<***reference-type***>> *(p. 351)* A wrapper class for the user-defined implementation of a **ContentFilter** (p. 950) 967
rti::core::policy::Database
<<***extension***>> *(p. 116)* Configures threads and resource limits that RTI Connext uses to control its internal database 969
dds::sub::DataReader< T >
<<***reference-type***>> *(p. 351)* Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **dds::sub::Subscriber** (p. 2410) 976
rti::core::status::DataReaderCacheStatus
<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask**↩ **::datareader_cache()** (p. 2386) 1046
rti::core::policy::DataReaderInstanceRemovalKind_def
Sets the kinds of instances that can be replaced when instance resource limits (**dds::core::policy**↩ **::ResourceLimits::max_instances** (p. 2206)) are reached 1053

dds::sub::DataReaderListener< T >
The **Listener** (p. 1637) to notify status changes for a **dds::sub::DataReader** (p. 976) 1055
rti::core::policy::DataReaderProtocol
<<***extension***>> *(p. 116)* Configures DataReader-specific aspects of the RTPS protocol 1059
rti::core::status::DataReaderProtocolStatus
<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::datareader_protocol()** (p. 2387) 1065
dds::sub::qos::DataReaderQos
<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::sub::DataReader** (p. 976) supports 1072
rti::core::policy::DataReaderResourceLimits
<<***extension***>> *(p. 116)* Configures the memory usage of a dds::pub::DataReader 1081
rti::core::DataReaderResourceLimitsInstanceReplacementSettings
<<***extension***>> *(p. 354)* How instances are replaced in the DataReader queue when resource limits are reached 1108
rti::sub::cond::DataReaderStatusConditionHandler< T >
Realization of a **functor handler** (p. 2134) that handles the status of a **dds::sub::DataReader** (p. 976) 1111
dds::core::policy::DataRepresentation
Contains the data representations supported by entities 1113
dds::sub::status::DataState
Describes the state of a sample and includes the information about the sample's **InstanceState** (p. 1615), **ViewState** (p. 2614), and **SampleState** (p. 2316) 1118
rti::sub::status::DataStateEx
<<***extension***>> (p. *354)* <<***value-type***>> *(p. 350)* An extended version of **dds::sub::status::DataState** (p. 1118) that also contains **StreamKind** (p. 2390) 1126
dds::core::policy::DataTag
Stores name-value (string) pairs that can be used to determine access permissions 1132
dds::pub::DataWriter< T >
<<***reference-type***>> *(p. 351)* Allows an application to publish data for a **dds::topic::Topic** (p. 2474) 1138
rti::core::status::DataWriterCacheStatus
<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::datawriter_cache()** (p. 2386) 1199
dds::pub::DataWriterListener< T >
The **Listener** (p. 1637) to notify status changes for a **dds::pub::DataWriter** (p. 1138) 1202
rti::core::policy::DataWriterProtocol
<<***extension***>> *(p. 116)* Configures aspects of an the RTPS protocol related to a **dds::pub::DataWriter** (p. 1138) 1210
rti::core::status::DataWriterProtocolStatus
<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::datawriter_protocol()** (p. 2386) 1219
dds::pub::qos::DataWriterQos
<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::pub::DataWriter** (p. 1138) supports 1226
rti::core::policy::DataWriterResourceLimits
<<***extension***>> *(p. 116)* Configures the memory usage of a **dds::pub::DataWriter** (p. 1138) . . 1235
rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_def
<<***extension***>> *(p. 354)* The enumeration for DataWriter Resource Limits 1249
rti::core::DataWriterShmemRefTransferModeSettings
<<***extension***>> *(p. 354)* Configures aspects of the shared memory reference transfer mode related to a DataWriter 1251
rti::core::policy::DataWriterTransferMode
<<***extension***>> *(p. 116)* Configures the transfer mode of a **dds::pub::DataWriter** (p. 1138) . . . 1253

dds::core::policy::Deadline
Expresses the maximum duration (deadline) within which an instance is expected to be updated . 1255
dds::core::policy::DestinationOrder
Controls the logical order of updates to the same instance by a **dds::pub::Publisher** (p. 1993) . . 1258
dds::core::policy::DestinationOrderKind_def
The definition of the **dds::core::safe_enum** (p. 2265) DestinationOrderKind 1263
rti::core::policy::DestinationOrderScopeKind_def
<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) DestinationOrder↩ ScopeKind 1264
rti::test::DisableMonitoringLibraryGuard
RAII scope guard to temporarily disable monitoring library 1265
rti::core::policy::Discovery
<<***extension***>> *(p. 116)* Configures entity discovery 1265
rti::core::policy::DiscoveryConfig
<<***extension***>> *(p. 116)* Configures the discovery mechanism 1271
rti::core::policy::DiscoveryConfigBuiltinChannelKindMask
<<***extension***>> *(p. 354)* A mask that selects the built-in channels to be used 1319
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask
<<***extension***>> *(p. 354)* A mask that selects the built-in discovery plugins to be used 1322
dds::domain::DomainParticipant
<<***reference-type***>> *(p. 351)* Container for all **dds::core::Entity** (p. 1520) objects . . 1326
rti::domain::DomainParticipantConfigParams
<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Input paramaters for creating a participant from xml configuration. It allows modification of some of the properties of the entities defined in the configuration 1371
dds::domain::qos::DomainParticipantFactoryQos
<<***value-type***>> *(p. 350)* Container of the QoS policies that do not apply to a specific entity 1378
dds::domain::DomainParticipantListener
The listener class for a **DomainParticipant** (p. 1326) 1382
rti::core::status::DomainParticipantProtocolStatus 1383
dds::domain::qos::DomainParticipantQos
<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::domain::DomainParticipant** (p. 1326) supports 1384
rti::core::policy::DomainParticipantResourceLimits
<<***extension***>> *(p. 116)* Configures the memory usage of certain **dds::domain::Domain↩ Participant** (p. 1326) resources 1391
dds::core::policy::Durability
Specifies whether a **dds::pub::DataWriter** (p. 1138) will store and deliver previously published data samples to late-joining **dds::sub::DataReader** (p. 976) entities 1433
dds::core::policy::DurabilityKind_def
The definition of the **dds::core::safe_enum** (p. 2265) DurabilityKind 1441
dds::core::policy::DurabilityService
Configures the external RTI Persistence Service used by persistent and transient DataWriters . . . 1443
dds::core::Duration
<<***value-type***>> *(p. 350)* Represents a time interval 1448
rti::topic::dynamic_type< TopicType >
Provides a DynamicType that represents an IDL-generated type 1461
dds::core::xtypes::DynamicData
<<***value-type***>> *(p. 350)* A data sample of any complex data type, which can be inspected and manipulated reflectively 1461
rti::core::xtypes::DynamicDataInfo
Contains information about a DynamicData sample 1494

rti::core::xtypes::DynamicDataMemberInfo
Contains information about a DynamicData member 1495
rti::core::xtypes::DynamicDataProperty
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Specifies the properties of a DynamicData object 1497
rti::core::xtypes::DynamicDataTypeSerializationProperty
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Configures aspects of the memory management in the serialization of **dds::core::xtypes::DynamicData** (p. 1461) samples 1500
dds::core::xtypes::DynamicType
<<***value-type***>> (p. *350*) Represents a runtime type 1504
rti::core::xtypes::DynamicTypePrintFormatProperty
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) A collection of attributes used to configure how DynamicTypes will be formatted when converted to strings 1509
rti::core::EndpointGroup
<<***extension***>> (p. *354*) Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum 1515
rti::topic::trust::EndpointTrustAlgorithmInfo
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Trust Plugins algorithm information associated with the discovered endpoint 1516
rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Trust Plugins interception algorithm information associated with the discovered endpoint 1517
rti::topic::trust::EndpointTrustProtectionInfo
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Trust Plugins Protection information associated with the discovered endpoint 1519
dds::core::Entity
<<***reference-type***>> (p. *351*) This is the abstract base class for all the DDS objects that support QoS policies, a listener and a status condition 1520
dds::core::policy::EntityFactory
Configures a **dds::core::Entity** (p. 1520) that acts as factory of other entities 1527
rti::core::policy::EntityName
<<***extension***>> (p. *116*) Assigns a name to a DomainParticipant, Publisher, Subscriber, DataWriter or DataReader 1530
dds::core::xtypes::EnumMember
<<***value-type***>> (p. *350*) Represents a **EnumType** (p. 1535) member 1533
dds::core::xtypes::EnumType
<<***value-type***>> (p. *350*) Represents an IDL `enum` type 1535
rti::test::EnvVarToken 1539
dds::core::Error
A generic, unspecified **Error** (p. 1539) 1539
rti::core::policy::Event
<<***extension***>> (p. *116*) Configures the thread in a DomainParticipant that handles timed events 1541
rti::core::status::EventCount< IntegerType >
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Encapsulates an event count containing a total count and an incremental count since the last time a status was read 1544
dds::core::Exception
The abstract base class for all of the DDS exceptions which may be thrown by the API 1546
rti::topic::ExpressionProperty
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Provides additional information about the filter expression passed to the writer_compile method of **rti::topic::WriterContentFilter** (p. 2652) . 1547
rti::topic::extensibility< TopicType >
<<***extension***>> (p. *354*) Indicates the extensibility kind of a topic-type 1550
dds::core::xtypes::ExtensibilityKind_def
The definition of the **dds::core::safe_enum** (p. 2265) ExtensibilityKind 1550

dds::core::external< T >
A managed reference to an object . . . 1551
dds::topic::Filter
Defines the filter to create a **ContentFilteredTopic** (p. 954) . . . 1558
rti::topic::FilterSampleInfo
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Provides meta information associated with the sample . . . 1562
rti::flat::FinalAlignedArrayOffset< ElementOffset, N >
Offset to an array of final elements . . . 1565
rti::flat::FinalArrayOffset< ElementOffset, N >
Offset to an array of final elements . . . 1567
rti::flat::FinalOffset< T >
The base class of all Offsets to a final struct type . . . 1568
rti::flat::FinalSequenceBuilder< ElementOffset >
Builds a sequence member of fixed-size elements . . . 1569
rti::flat::flat_type_traits< T >
Given a **Sample** (p. 2274), an Offset or a Builder, it allows obtaining the other types . . . 1571
rti::pub::FlowController
<<***extension***>> (p. *354*) <<***reference-type***>> (p. *351*) A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **dds::pub::DataWriter** (p. 1138) instances are allowed to write data . . . 1572
rti::pub::FlowControllerProperty
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Configures a **FlowController** (p. 1572) . . . 1577
rti::pub::FlowControllerSchedulingPolicy_def
<<***extension***>> (p. *354*) Kinds of flow controller shceduling policy . . . 1581
rti::pub::FlowControllerTokenBucketProperty
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Configures a **FlowController** (p. 1572) based on a tocken-bucket mechanism . . . 1583
Foo
An example topic-type . . . 1588
dds::sub::GenerationCount
<<***value-type***>> (p. *350*) . . . 1589
dds::core::policy::GroupData . . . 1591
dds::core::cond::GuardCondition
<<***reference-type***>> (p. *351*) A condition whose trigger value is under the control of the application 1594
rti::core::Guid
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Class for GUID (Global Unique Identifier) representation
1596
rti::util::heap_monitoring::HeapMonitoringParams
Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot . . . 1599
dds::core::policy::History
Specifies how much historical data a **dds::pub::DataWriter** (p. 1138) and a **dds::sub::DataReader** (p. 976) can store . . . 1602
dds::core::policy::HistoryKind_def
The definition of the **dds::core::safe_enum** (p. 2265) HistoryKind . . . 1606
rti::core::policy::IgnoredEntityReplacementKind_def
<<***extension***>> (p. *354*) The enumeration for **DomainParticipantResourceLimits** (p. 1391) ignored entity replacement kinds . . . 1607
dds::core::IllegalOperationError
Indicates that an operation was called under improper circumstances . . . 1608
dds::core::ImmutablePolicyError
Indicates that the application attempted to modify an immutable QoS policy . . . 1609

dds::core::InconsistentPolicyError
Indicates that the application specified a set of QoS policies that are not consistent with each other 1610
dds::core::status::InconsistentTopicStatus
Information about the status **dds::core::status::StatusMask::inconsistent_topic()** (p. 2379) . . 1611
dds::core::InstanceHandle
<<***value-type***>> *(p. 350)* Handle to identify different instances of the same **dds::topic::Topic** (p. 2474) of a certain type 1612
dds::sub::status::InstanceState
Indicates if the samples are from a live **dds::pub::DataWriter** (p. 1138) or not 1615
dds::core::InvalidArgumentError
Indicates that the application passed an illegal parameter value into an operation 1619
dds::core::InvalidDowncastError
Indicates that a downcast was incorrect 1620
rti::core::status::InvalidLocalAdvanceNoticeStatus
<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::invalid_local_advance_notice()** (p. 2388) 1621
dds::topic::is_topic_type< T >
Trait that indicates if a type is suitable to be the type of a **dds::topic::Topic** (p. 2474) 1622
rti::request::IsReplyRelatedPredicate< T >
Functor to match replies by their related request 1622
rti::sub::IsValidData< T >
<<***extension***>> *(p. 354)* A functor that returns true when a sample has valid data 1624
dds::core::KeyedBytesTopicType
Built-in type consisting of a variable-length array of opaque bytes and a string that is the key . . . 1626
dds::core::KeyedStringTopicType
Built-in type consisting of a string payload and a second string that is the key 1629
dds::core::policy::LatencyBudget
Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications 1631
rti::config::LibraryVersion
<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* The version of a single library shipped as part of an RTI Connext distribution 1634
dds::core::policy::Lifespan
Specifies how long the data written by a **dds::pub::DataWriter** (p. 1138) is considered valid 1635
Listener
Entity listeners 1637
rti::core::ListenerBinder< Entity, Listener >
<<***reference-type***>> *(p. 351)* Automatically manages the association of an Entity and a **Listener** (p. 1637) 1641
dds::core::policy::Liveliness
Specifies and configures the mechanism that allows **dds::sub::DataReader** (p. 976)'s to detect when **dds::pub::DataWriter** (p. 1138)'s become disconnected 1645
dds::core::status::LivelinessChangedStatus
Information about the status **dds::core::status::StatusMask::liveliness_changed()** (p. 2383) . . 1652
dds::core::policy::LivelinessKind_def
The definition of the **dds::core::safe_enum** (p. 2265) LivelinessKind 1654
dds::core::status::LivelinessLostStatus
Information about the status **dds::core::status::StatusMask::liveliness_lost()** (p. 2383) 1655
rti::core::xtypes::LoanedDynamicData
<<***move-only-type***>> *(p. 353)* Gives temporary access to a member of another DynamicData object 1656
rti::sub::LoanedSample< T >
The element type of a **dds::sub::LoanedSamples** (p. 1663) collection 1659

dds::sub::LoanedSamples< T >
<<***move-only-type***>> *(p. 353)* Provides temporary access to a collection of samples (data and info) from a **DataReader** (p. 976) 1663
rti::core::Locator
<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports . . 1673
rti::core::policy::LocatorFilter
<<***extension***>> *(p. 116)* Configures how the **dds::topic::PublicationBuiltinTopicData** (p. 1977) reports the configuration of a **MultiChannel** (p. 1749) DataWriter 1676
rti::core::LocatorFilterElement
<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Specifies the configuration of an individual channel within a MultiChannel DataWriter 1678
rti::core::LocatorKind_def
The definition of the **dds::core::safe_enum** (p. 2265) LocatorKind 1681
rti::config::LogCategory_def
The definition of the **dds::core::safe_enum** (p. 2265) LogCategory 1682
rti::config::Logger
The singleton type used to configure RTI Connext logging 1683
rti::config::LogLevel_def
The definition of the **dds::core::safe_enum** (p. 2265) LogLevel; 1692
rti::config::LogMessage
A log message's input parameter to an output handler, including the text and additional information 1693
rti::core::LongDouble
<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Encapsulates an IDL `long double` . . . 1696
dds::sub::DataReader< T >::ManipulatorSelector
A **Selector** (p. 2320) class enabling the streaming API 1697
rti::sub::ManipulatorSelector< T > 1700
dds::core::xtypes::Member
<<***value-type***>> *(p. 350)* Represents a **StructType** (p. 2400) member 1700
rti::core::policy::Monitoring
<<***extension***>> *(p. 116)* Configures the use of the RTI **Monitoring** (p. 1706) Library 2.0 to collect and distribute RTI Connext telemetry data 1706
rti::core::MonitoringDedicatedParticipantSettings
<<***extension***>> *(p. 354)* Configures the usage of a dedicated **dds::domain::DomainParticipant** (p. 1326) to distribute the RTI Connext application telemetry data 1712
rti::core::MonitoringDistributionSettings
<<***extension***>> *(p. 354)* Configures the distribution of telemetry data 1716
rti::core::MonitoringEventDistributionSettings
<<***extension***>> *(p. 354)* Configures the distribution of event metrics 1722
rti::core::MonitoringLoggingDistributionSettings
<<***extension***>> *(p. 354)* Configures the distribution of log messages 1727
rti::core::MonitoringLoggingForwardingSettings
<<***extension***>> *(p. 354)* Configures the forwarding levels of log messages for the different **rti::config::LogFacility** (p. 459) 1733
rti::core::MonitoringMetricSelection
<<***extension***>> *(p. 354)* Configures event and periodic metrics collection and distribution for a specific set of observable resources 1736
rti::core::MonitoringPeriodicDistributionSettings
<<***extension***>> *(p. 354)* Configures the distribution of periodic metrics 1740
rti::core::MonitoringTelemetryData
<<***extension***>> *(p. 354)* Configures the telemetry data that will be distributed 1744
rti::core::policy::MulticastMapping
<<***extension***>> *(p. 116)* Type representing a list of multicast mapping elements 1747

rti::core::policy::MultiChannel
<<***extension***>> *(p. 116)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data . . . 1749
rti::flat::MutableArrayBuilder< ElementBuilder, N >
Builds an array member of variable-size elements . . . 1753
rti::flat::MutableArrayOffset< ElementOffset, N >
Offset to an array of variable-size elements . . . 1755
rti::flat::MutableOffset
The base class of all Offsets to a final struct type . . . 1756
rti::flat::MutableSequenceBuilder< ElementBuilder >
Builds a sequence member of variable-size elements . . . 1757
MyFlatFinalOffset
Represents the Offset to an arbitrary user-defined FlatData final IDL struct . . . 1759
MyFlatMutableBuilder
Represents the Builder for an arbitrary user-defined mutable type . . . 1763
MyFlatMutableOffset
Represents the Offset to an arbitrary user-defined FlatData mutable IDL struct . . . 1770
MyFlatUnionBuilder
Represents the Builder for an arbitrary user-defined mutable IDL union . . . 1777
MyFlatUnionOffset
Represents the Offset to an arbitrary user-defined FlatData mutable IDL union . . . 1780
NDDS_Transport_Address_t
Addresses are stored individually as network-ordered bytes . . . 1784
NDDS_Transport_Interface_t
Storage for the description of a network interface used by a Transport Plugin . . . 1785
NDDS_Transport_Property_t
Base configuration structure that must be inherited by derived Transport Plugin classes . . . 1786
NDDS_Transport_Shmem_Property_t
Subclass of **NDDS_Transport_Property_t** (p. 1786) allowing specification of parameters that are specific to the shared-memory transport . . . 1794
NDDS_Transport_UDP_WAN_CommPortsMappingInfo
Type for storing UDP WAN communication ports . . . 1797
NDDS_Transport_UDPv4_Property_t
Configurable IPv4/UDP Transport-Plugin properties . . . 1798
NDDS_Transport_UDPv4_WAN_Property_t
Configurable IPv4/UDP WAN Transport-Plugin properties . . . 1809
NDDS_Transport_UDPv6_Property_t
Configurable IPv6/UDP Transport-Plugin properties . . . 1819
NDDS_Transport_UUID
Unequivocally identifies a transport plugin instance . . . 1829
rti::util::network_capture::NetworkCaptureParams
<<***extension***>> *(p. 354)* Input parameters for starting Network Capture . . . 1829
rti::topic::no_compile_data_t
The type to specify as the CompileData template parameter to your **ContentFilter** (p. 950) if your compile function does not return any data . . . 1835
rti::queuing::NoMatchingQueueException
The entity lost matching with all the Queuing Service . . . 1835
rti::core::cond::NoOpAsyncWaitSetListener
A convenience implementation of **AsyncWaitSetListener** (p. 856) where all methods are overridden to do nothing . . . 1836
dds::sub::NoOpDataReaderListener< T >
A convenience implementation of **DataReaderListener** (p. 1055) where all methods are overridden to do nothing . . . 1838

dds::pub::NoOpDataWriterListener< T >
A convenience implementation of **DataWriterListener** (p. 1202) where all methods are overridden to do nothing . . . 1841
dds::domain::NoOpDomainParticipantListener
A convenience implementation of **DomainParticipantListener** (p. 1382) where all methods are overridden to do nothing . . . 1845
dds::pub::NoOpPublisherListener
A convenience implementation of **PublisherListener** (p. 2006) where all methods are overridden to do nothing . . . 1852
rti::queuing::NoOpQueueConsumerListener< T >
A listener with an empty implementation of all methods . . . 1856
rti::queuing::NoOpQueueProducerListener< T >
A listener with an empty implementation of all methods . . . 1858
rti::queuing::NoOpQueueReplierListener< TReq, TRep >
A listener with an empty implementation of all methods . . . 1859
rti::queuing::NoOpQueueRequesterListener< TReq, TRep >
A listener with an empty implementation of all methods . . . 1861
dds::sub::NoOpSubscriberListener
A convenience implementation of **SubscriberListener** (p. 2422) where all methods are overridden to do nothing . . . 1864
dds::topic::NoOpTopicListener< T >
A convenience implementation of **TopicListener** (p. 2508) where all methods are overridden to do nothing . . . 1867
dds::core::NotAllowedBySecurityError
Indicates that an operation on the DDS API fails because the security plugins do not allow it . . . 1868
dds::core::NotEnabledError
A **NotEnabledError** (p. 1869) is thrown when an operation is invoked on a **dds::core::Entity** (p. 1520) that is not yet enabled . . . 1869
dds::core::NullReferenceError
Indicates an attempt to access a null object . . . 1870
rti::core::xtypes::octet_tag_t
*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* A special template type to obtain an IDL octet DynamicType from primitive_type . . . 1871
dds::core::status::OfferedDeadlineMissedStatus
Information about the status **dds::core::status::StatusMask::offered_deadline_missed()** (p. 2379) . . . 1871
dds::core::status::OfferedIncompatibleQosStatus
Information about the status **dds::core::status::StatusMask::offered_incompatible_qos()** (p. 2380) . . . 1873
rti::flat::OffsetBase
Base class of all Offset types . . . 1875
dds::core::optional< T >
*<<**value-type**>> (p. 350)* Represents an object that may not contain a valid value . . . 1879
rti::core::optional_value< T >
*<<**extension**>> (p. 354)* Represents a value that can be initialized or not . . . 1892
OsapiFile . . . 1898
dds::core::OutOfResourcesError
Indicates that RTI Connext ran out of the resources needed to complete the operation . . . 1898
dds::core::policy::Ownership
Specifies whether it is allowed for multiple **dds::pub::DataWriter** (p. 1138)'s to write the same instance of the data and if so, how these modifications should be arbitrated . . . 1899
dds::core::policy::OwnershipKind_def
The definition of the **dds::core::safe_enum** (p. 2265) OwnershipKind . . . 1905

dds::core::policy::OwnershipStrength
Specifies the value of the strength used to arbitrate among multiple **dds::pub::DataWriter** (p. 1138) objects that attempt to modify the same instance of a data type (identified by its **dds::topic::Topic** (p. 2474) and key) . . . 1906
dds::topic::ParticipantBuiltinTopicData
Entry created when a **dds::domain::DomainParticipant** (p. 1326) is discovered . . . 1908
rti::test::ParticipantFactoryGuard
RAII guard for managing the DomainParticipantFactory lifecycle . . . 1916
rti::topic::trust::ParticipantTrustAlgorithmInfo
*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins algorithm information associated with the discovered DomainParticipant . . . 1916
rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo
*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins interception algorithm information associated with the discovered DomainParticipant . . . 1917
rti::topic::trust::ParticipantTrustKeyEstablishmentAlgorithmInfo
*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant . . . 1919
rti::topic::trust::ParticipantTrustProtectionInfo
*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins Protection information associated with the discovered DomainParticipant . . . 1920
rti::topic::trust::ParticipantTrustSignatureAlgorithmInfo
*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins signature algorithm information associated with the discovered DomainParticipant . . . 1922
dds::core::policy::Partition
Set of strings that introduces logical partitions in **dds::domain::DomainParticipant** (p. 1326), **dds::pub::Publisher** (p. 1993), or **dds::sub::Subscriber** (p. 2410) entities . . . 1923
rti::core::PersistentStorageSettings
*<<**extension**>> (p. 354)* Configures the persistent storage settings for durable writer history and durable reader state . . . 1927
rti::core::pointer< T > . . . 1938
dds::core::policy::policy_id< Policy >
Obtains the QosPolicyId of a QoS Policy . . . 1940
dds::core::policy::policy_name< Policy >
Obtains the policy name . . . 1941
dds::core::PreconditionNotMetError
A **PreconditionNotMetError** (p. 1941) is thrown when a pre-condition for the operation was not met 1941
dds::core::policy::Presentation
Specifies how the samples representing changes to data instances are presented to a subscribing application . . . 1942
dds::core::policy::PresentationAccessScopeKind_def
The definition of the **dds::core::safe_enum** (p. 2265) PresentationAccessScopeKind . . . 1950
rti::flat::PrimitiveArrayOffset< T, N >
Offset to an array of primitive elements . . . 1951
rti::flat::PrimitiveConstOffset< T >
A const Offset to an optional primitive member . . . 1952
rti::flat::PrimitiveOffset< T >
An Offset to an optional primitive member . . . 1953
rti::flat::PrimitiveSequenceBuilder< T >
Builds a sequence of primitive members . . . 1955
rti::flat::PrimitiveSequenceOffset< T >
Offset to a sequence of primitive elements . . . 1958
dds::core::xtypes::PrimitiveType
*<<**value-type**>> (p. 350)* Represents an IDL primitive type . . . 1959

rti::config::PrintFormat_def
The definition of the **dds::core::safe_enum** (p. 2265) PrintFormat . . . 1960
rti::topic::PrintFormatKind_def
The definition of the **dds::core::safe_enum** (p. 2265) PrintFormatKind . . . 1961
rti::topic::PrintFormatProperty
*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* A collection of attributes used to configure how data samples will be formatted when converted to a string . . . 1962
rti::core::ProductVersion
*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Represents the current version of RTI Connext . . . 1967
rti::core::policy::Property
*<<**extension**>> (p. 116)* Stores key/value string pairs that can configure certain parameters of RTI Connext not exposed through QoS policies. It can also store and propagate through the discovery mechanism application-specific information associated to a **dds::core::Entity** (p. 1520) . . . 1969
rti::core::ProtocolVersion
*<<**extension**>> (p. 354)* Represents the current version of RTI Connext . . . 1975
dds::topic::PublicationBuiltinTopicData
Entry created when a **dds::pub::DataWriter** (p. 1138) is discovered in association with its **dds::pub::Publisher** (p. 1993) . . . 1977
dds::core::status::PublicationMatchedStatus
Information about the status **dds::core::status::StatusMask::publication_matched()** (p. 2384) . . . 1990
dds::pub::Publisher
*<<**reference-type**>> (p. 351)* A publisher is the object responsible for the actual dissemination of publications . . . 1993
dds::pub::PublisherListener
The listener to notify status changes for a **dds::pub::Publisher** (p. 1993) . . . 2006
dds::pub::qos::PublisherQos
*<<**value-type**>> (p. 350)* Container of the QoS policies that a **dds::pub::Publisher** (p. 1993) supports . . . 2007
rti::core::policy::PublishMode
*<<**extension**>> (p. 116)* Specifies whether a **dds::pub::DataWriter** (p. 1138) sends data synchronously or asynchronously . . . 2013
rti::core::policy::PublishModeKind_def
*<<**extension**>> (p. 354)* The enumeration for **PublishMode** (p. 2013) kinds . . . 2018
rti::core::qos_print_all_t
A tag type that selects the to_string overload that prints all the values of a Qos object . . . 2019
dds::core::policy::QosPolicyCount
*<<**value-type**>> (p. 350)* Holds a counter for a QosPolicyId . . . 2020
rti::core::QosPrintFormat
*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* A collection of attributes used to configure how QoS will be formatted when converted to strings . . . 2021
dds::core::QosProvider
*<<**reference-type**>> (p. 351)* The **QosProvider** (p. 2025) class provides a way for a user to control and access the XML QoS profiles that are loaded by RTI Connext . . . 2025
rti::core::QosProviderParams
*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Configure options that control the way that XML documents containing QoS profiles are loaded by a **dds::core::QosProvider** (p. 2025) . . . 2049
dds::sub::Query
*<<**value-type**>> (p. 350)* Encapsulates a query for a **dds::sub::cond::QueryCondition** (p. 2059) 2053
dds::sub::cond::QueryCondition
*<<**reference-type**>> (p. 351)* Specialized **ReadCondition** (p. 2133) that allows applications to also specify a filter on the data available in a **dds::sub::DataReader** (p. 976) . . . 2059
rti::queuing::QueueConsumer< T >
Allows you to receive samples from a SharedReaderQueue . . . 2063

rti::queuing::QueueConsumerListener< T >
Called when certain events occur in a **QueueConsumer** (p. 2063) . . . 2075
rti::queuing::QueueConsumerParams
Contains the parameters for creating a **QueueConsumer** (p. 2063) . . . 2077
rti::queuing::QueueEntityParams< ActualEntity >
A parent class for all queue parameter classes . . . 2079
rti::queuing::QueueProducer< T >
Allows you to send samples to a specific SharedReaderQueue hosted by Queuing Service . . . 2080
rti::queuing::QueueProducerListener< T >
Called when certain events occur in a **QueueProducer** (p. 2080) . . . 2090
rti::queuing::QueueProducerParams
Contains the parameters for creating a **QueueProducer** (p. 2080) . . . 2092
rti::queuing::QueueReplier< TReq, TRep >
Allows receiving requests and sending replies . . . 2094
rti::queuing::QueueReplierListener< TReq, TRep >
Called when certain events occur in a **QueueReplier** (p. 2094) . . . 2107
rti::queuing::QueueReplierParams
Contains the parameters for creating a **QueueReplier** (p. 2094) . . . 2109
rti::queuing::QueueRequester< TReq, TRep >
Allows sending requests and receiving replies . . . 2111
rti::queuing::QueueRequesterListener< TReq, TRep >
Called when certain events occur in a **QueueRequester** (p. 2111) . . . 2126
rti::queuing::QueueRequesterParams
Contains the parameters for creating a **QueueRequester** (p. 2111) . . . 2128
dds::sub::Rank
<<***value-type***>> *(p. 350)* Contains the sample and generation ranks of a data-sample . . . 2130
dds::sub::cond::ReadCondition
<<***reference-type***>> *(p. 351)* Condition specifically dedicated to read operations and attached to one **dds::sub::DataReader** (p. 976) . . . 2133
dds::core::policy::ReaderDataLifecycle
Controls how a DataReader manages the lifecycle of the data that it has received . . . 2137
dds::sub::ReadModeDummyType . . . 2144
rti::core::policy::ReceiverPool
<<***extension***>> *(p. 116)* Configures threads that RTI Connext uses to receive and process data from the transport modules (such as UDP) . . . 2144
dds::core::Reference< DELEGATE >
Base class for all reference types . . . 2149
dds::core::policy::Reliability
Indicates the level of reliability in sample delivered that a **dds::pub::DataWriter** (p. 1138) offers or a **dds::sub::DataReader** (p. 976) requests . . . 2149
dds::core::policy::ReliabilityKind_def
The definition of the **dds::core::safe_enum** (p. 2265) ReliabilityKind . . . 2156
rti::core::status::ReliableReaderActivityChangedStatus
<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::reliable_reader_activity_changed()** (p. 2385) . . . 2157
rti::core::status::ReliableWriterCacheChangedStatus
<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385) . . . 2159
dds::rpc::RemoteInvalidArgumentError
Thrown when a Client provides an invalid argument to a Service operation . . . 2162
dds::rpc::RemoteOutOfResourcesError
Thrown when the Service runs out of resources to fulfill a Client's operation . . . 2163

rti::core::policy::RemoteParticipantPurgeKind_def
<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) RemoteParticipantPurgeKind 2164
dds::rpc::RemoteUnknownExceptionError
Thrown when an unknown exception occurs in the Service 2165
dds::rpc::RemoteUnknownOperationError
Thrown when a Client calls an operation that doesn't exist in the Service 2166
dds::rpc::RemoteUnsupportedOperationError
Thrown when a Client calls an operation that is not supported by the Service 2167
rti::request::Replier< RequestType, ReplyType >
<<***reference-type***>> *(p. 351)* Allows receiving requests and sending replies 2168
rti::request::ReplierListener< RequestType, ReplyType >
Called when a **Replier** (p. 2168) has new available requests 2179
rti::request::ReplierParams
Contains the parameters for creating a **rti::request::Replier** (p. 2168) 2180
dds::core::status::RequestedDeadlineMissedStatus
Information about the status **dds::core::status::StatusMask::requested_deadline_missed()** (p. 2380) 2184
dds::core::status::RequestedIncompatibleQosStatus
Information about the status **dds::core::status::StatusMask::requested_incompatible_qos()** (p. 2381) 2185
rti::request::Requester< RequestType, ReplyType >
<<***reference-type***>> *(p. 351)* Allows sending requests and receiving replies 2187
rti::request::RequesterParams
Contains the parameters for creating a **rti::request::Requester** (p. 2187) 2199
dds::core::policy::ResourceLimits
Controls the memory usage of **dds::pub::DataWriter** (p. 1138) or a **dds::sub::DataReader** (p. 976) 2203
rti::core::Result< T >
A result from an operation that doesn't throw exceptions containing a return code and (if successful) a value 2209
rpc_example::RobotControl
The synchronous interface generated from the **RobotControl** (p. 97) IDL service 2213
rpc_example::RobotControlAsync
The asynchronous interface derived from the **RobotControl** (p. 2213) service 2215
rpc_example::RobotControlClient
<<***reference-type***>> *(p. 351)* Allows client applications to make remote function calls 2217
rti::config::compliance::RpcMask
XTypes compliance mask 2218
rti::core::RtpsReliableReaderProtocol
<<***extension***>> *(p. 354)* Configures aspects of the RTPS protocol related to a reliable DataReader 2221
rti::core::policy::RtpsReliableWriterProtocol
<<***extension***>> *(p. 116)* Configures aspects of an RTPS reliable writer 2229
rti::core::policy::RtpsReservedPortKindMask
<<***extension***>> *(p. 354)* Mask of reserved ports 2255
rti::core::RtpsWellKnownPorts
<<***extension***>> *(p. 354)* Configures the mapping of the RTPS well-known ports 2258
dds::core::safe_enum< def, inner >
<<***value-type***>> *(p. 350)* Provides a safe, scoped enumeration based on def::type 2265
dds::sub::Sample< T >
<<***value-type***>> *(p. 350)* This class encapsulate the data and meta-data associated with DDS samples 2270
rti::flat::Sample< OffsetType >
The generic definition of FlatData topic-types 2274

rti::core::SampleFlag
<<***extension***>> *(p. 354)* A set of flags that can be associated with a sample 2278
rti::core::SampleIdentity
<<***extension***>> (p. *354)* <<***value-type***>> *(p. 350)* A **SampleIdentity** (p. 2282) defines a pair (Virtual Writer **Guid** (p. 1596), **SequenceNumber** (p. 2335)) that uniquely identifies a sample within a DDS domain and a Topic 2282
dds::sub::SampleInfo
<<***value-type***>> *(p. 350)* Information that accompanies each sample received by a **DataReader** (p. 976) 2285
rti::sub::SampleIterator< T >
A random-access iterator of **LoanedSample** (p. 1659) 2296
rti::core::status::SampleLostState
<<***extension***>> (p. *354)* Reasons why a sample was lost 2296
dds::core::status::SampleLostStatus
Information about the status **dds::core::status::StatusMask::sample_lost()** (p. 2381) 2303
rti::sub::SampleProcessor
<<***extension***>> (p. *354)* <<***reference-type***>> *(p. 351)* Utility to read and process the data samples that one or more DataReaders receive using a sample handler 2305
dds::core::status::SampleRejectedState
Reasons why a sample was rejected 2310
dds::core::status::SampleRejectedStatus
Information about the status **dds::core::status::StatusMask::sample_rejected()** (p. 2382) 2315
dds::sub::status::SampleState
Indicates whether or not a sample has ever been read 2316
rti::config::ScopedLoggerVerbosity
Changes the logger verbosity temporarily during the scope of a variable 2319
dds::sub::DataReader< T >::Selector
Used by the **DataReader** (p. 976) to compose read and take operations 2320
rti::flat::SequenceIterator< E, OffsetKind >
Iterator for collections of Offsets 2328
rti::core::SequenceNumber
<<***extension***>> (p. *354)* <<***value-type***>> *(p. 350)* A class representing the DDS 64-bit Sequence Number 2335
rti::flat::SequenceOffset< ElementOffset >
Offset to a sequence of non-primitive elements 2343
dds::core::xtypes::SequenceType
<<***value-type***>> (p. *350)* Represents an IDL `sequence` type 2344
dds::rpc::Server
<<***reference-type***>> *(p. 351)* Provides the execution environment for one or more **Service↩ Endpoint** (p. 2354) 2346
dds::rpc::ServerParams
<<***value-type***>> *(p. 350)* The parameters used to configure a **Server** (p. 2346) 2348
rti::core::policy::Service
<<***extension***>> *(p. 116)* Indicates if an Entity is associated with a service or tool and if so, which one 2350
dds::rpc::ServiceEndpoint< Dispatcher >
Manages the DDS entities required to receive function calls and send the return values 2354
rti::core::policy::ServiceKind_def
<<***extension***>> (p. *354)* The definition of the **dds::core::safe_enum** (p. 2265) ServiceKind . . . 2357
rti::topic::ServiceRequest
<<***extension***>> (p. *354)* <<***value-type***>> *(p. 350)* A request coming from one of the built-in services 2358

rti::core::status::ServiceRequestAcceptedStatus
<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::service_request_accepted()** (p. 2388) . . . 2360
rti::core::ServiceRequestId_def
The definition of the rti::core::safe_enum ServiceRequestId . . . 2361
dds::sub::SharedSamples< T, DELEGATE >
<<***reference-type***>> *(p. 351)* A sharable and container-safe version of **LoanedSamples** (p. 1663) 2362
rti::sub::SharedSamples< T >
Provides access to a collection of middleware-loaned samples . . . 2367
rti::request::SimpleReplier< RequestType, ReplyType >
<<***reference-type***>> *(p. 351)* A callback-based replier . . . 2368
rti::util::heap_monitoring::SnapshotContentFormat_def
Bitmap used to decide which information of the snapshot will be displayed . . . 2370
rti::util::heap_monitoring::SnapshotOutputFormat_def
Specify the format of the output of the snapshot. RTI Connext . . . 2371
dds::core::cond::StatusCondition
<<***reference-type***>> *(p. 351)* A condition associated with each **dds::core::Entity** (p. 1520) . . . 2372
dds::core::status::StatusMask
A std::bitset (list) of statuses . . . 2375
rti::util::StreamFlagSaver . . . 2390
rti::sub::status::StreamKind
<<***extension***>> *(p. 354)* Indicates which stream to read from: live stream, topic-query stream or both . . . 2390
rti::flat::StringBuilder
Builds a string . . . 2392
rti::flat::StringOffset
Offset to a string . . . 2394
dds::core::StringTopicType
Built-in type consisting of a single character string . . . 2395
dds::core::xtypes::StringType
<<***value-type***>> *(p. 350)* Represents an IDL `string` type . . . 2399
dds::core::xtypes::StructType
<<***value-type***>> *(p. 350)* Represents an IDL `struct` type . . . 2400
dds::sub::Subscriber
<<***reference-type***>> *(p. 351)* A subscriber is the object responsible for actually receiving data from a subscription . . . 2410
dds::sub::SubscriberListener
The listener to notify status changes for a **dds::sub::Subscriber** (p. 2410) . . . 2422
dds::sub::qos::SubscriberQos
<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::sub::Subscriber** (p. 2410) supports . . . 2423
dds::topic::SubscriptionBuiltinTopicData
Entry created when a **dds::sub::DataReader** (p. 976) is discovered in association with its **dds::sub::Subscriber** (p. 2410) . . . 2428
dds::core::status::SubscriptionMatchedStatus
Information about the status **dds::core::status::StatusMask::subscription_matched()** (p. 2384) 2439
dds::pub::SuspendedPublication
<<***value-type***>> *(p. 350)* Indicates that the application is about to make multiple modifications using several **dds::pub::DataWriter** (p. 1138)'s belonging to the same **dds::pub::Publisher** (p. 1993) 2441
rti::core::policy::SystemResourceLimits
<<***extension***>> *(p. 116)* Configures resources that RTI Connext uses . . . 2443
dds::core::TEntityQos< DELEGATE >
Acts as a container for Qos policies allowing to set and retrieve all the policies of an entity as a unit 2446

rti::core::ThreadSettings
<<***extension***>> *(p. 354)* The properties of a thread of execution 2449
rti::core::ThreadSettingsCpuRotationKind_def
Determines how **rti::core::ThreadSettings::cpu_list** (p. 2451) affects processor affinity for thread-related QoS policies that apply to multiple threads 2453
rti::core::ThreadSettingsKindMask
<<***extension***>> *(p. 354)* A collection of flags used to configure threads of execution 2454
dds::core::Time
<<***extension***>> *(p. 354)* Represents a point in time 2457
dds::core::policy::TimeBasedFilter
Allows a **dds::sub::DataReader** (p. 976) to indicate that it is not interested in all the sample updates that occur within a time period 2469
dds::core::TimeoutError
Indicates that an operation has timed out 2473
dds::topic::Topic< **T** >
<<***reference-type***>> *(p. 351)* **Topic** (p. 2474) is the most basic description of the data to be published and subscribed 2474
rti::topic::topic_type_disabled_copy< **TopicType** >
<<***extension***>> *(p. 354)* Indicates whether a TopicType is uncopyable 2492
rti::topic::topic_type_has_external_members< **TopicType** >
<<***extension***>> *(p. 354)* Indicates if a topic type contains directly or indirectly IDL external members 2492
dds::topic::topic_type_name< **T** >
Provides the name of a topic-type 2493
dds::topic::topic_type_support< **T** >
Provides convenience operations for a topic-type 2493
dds::topic::TopicBuiltinTopicData
Entry created when a **dds::topic::Topic** (p. 2474) object is discovered 2493
dds::core::policy::TopicData 2500
dds::topic::TopicDescription< **T** >
Abstract base class of **Topic** (p. 2474) and **ContentFilteredTopic** (p. 954) 2503
dds::topic::TopicInstance< **T** >
Encapsulates a sample and its associated instance handle 2505
dds::topic::TopicListener< **T** >
The listener to notify status changes for a **dds::topic::Topic** (p. 2474) 2508
dds::topic::qos::TopicQos
<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::topic::Topic** (p. 2474) supports 2509
rti::sub::TopicQuery
<<***extension***>> *(p. 354)* <<***reference-type***>> *(p. 351)* Allows a **dds::sub::DataReader** (p. 976) to query the sample cache of its matching dds::pub::DataWriters 2516
rti::sub::TopicQueryData
<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Provides information about a **TopicQuery** (p. 2516) 2520
rti::core::policy::TopicQueryDispatch
<<***extension***>> *(p. 116)* Configures the ability of a **dds::pub::DataWriter** (p. 1138) to publish samples in response to a **rti::sub::TopicQuery** (p. 2516) 2522
rti::sub::TopicQuerySelection
<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Specifies the data query that defines a **TopicQuery** (p. 2516) 2526
rti::sub::TopicQuerySelectionKind_def
The definition of the **dds::core::safe_enum** (p. 2265) **rti::sub::TopicQuerySelectionKind** (p. 71) 2528
rti::util::network_capture::TrafficKindMask
<<***extension***>> *(p. 354)* Mask indicating the traffic direction to capture 2529

TransportAllocationSettings_t
Allocation settings used by various internal buffers . . . 2533
rti::core::policy::TransportBuiltin
<<***extension***>> *(p. 116)* Specifies which built-in transports to use . . . 2533
rti::core::policy::TransportBuiltinMask
<<***extension***>> *(p. 354)* Mask that specifies which built-in transports are used . . . 2537
rti::core::TransportClassId_def
The definition of the **dds::core::safe_enum** (p. 2265) TransportClassId . . . 2540
rti::core::TransportInfo
<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Contains the class id and message max size of an installed transport . . . 2541
rti::core::policy::TransportMulticast
<<***extension***>> *(p. 116)* Specifies the multicast address on which a **dds::sub::DataReader** (p. 976) wants to receive its data and other settings . . . 2543
rti::core::policy::TransportMulticastKind_def
<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) TransportMulticastKind . . . 2547
rti::core::policy::TransportMulticastMapping
<<***extension***>> *(p. 116)* Specifies a list of topic_expressions and multicast addresses that can be used by an Entity with a specific topic name to receive data . . . 2547
rti::core::TransportMulticastSettings
<<***extension***>> *(p. 354)* Represents a list of multicast locators . . . 2549
dds::core::policy::TransportPriority
Allows applications to take advantage of transports capable of sending messages with different priorities . . . 2552
rti::core::policy::TransportSelection
<<***extension***>> *(p. 116)* Specifies the transports that a **dds::pub::DataWriter** (p. 1138) or a **dds::sub::DataReader** (p. 976) may use to send or receive data . . . 2554
rti::core::policy::TransportUnicast
<<***extension***>> *(p. 116)* Specifies a subset of transports and a port number that can be used by a **dds::core::Entity** (p. 1520) to receive data . . . 2556
rti::core::TransportUnicastSettings
<<***extension***>> *(p. 354)* Represents a list of unicast locators . . . 2559
rti::topic::trust::TrustAlgorithmRequirements
<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Type to describe Trust Plugins algorithm requirements . . . 2562
dds::core::policy::TypeConsistencyEnforcement
Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it . . . 2563
dds::core::policy::TypeConsistencyEnforcementKind_def
The definition of the **dds::core::safe_enum** (p. 2265) TypeConsistencyEnforcementKind . . . 2570
dds::core::xtypes::TypeKind_def
The definition of TypeKind . . . 2571
rti::core::policy::TypeSupport
<<***extension***>> *(p. 116)* Allows attaching application-specific information to a **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976) that is passed to the serilization and deserialization routines . . . 2574
dds::core::xtypes::UnidimensionalCollectionType
<<***value-type***>> *(p. 350)* The base class of collection types with only one dimension . . . 2576
rti::flat::UnionBuilder< Discriminator >
Base class of builders for user-defined mutable unions . . . 2577
dds::core::xtypes::UnionMember
<<***value-type***>> *(p. 350)* Represents a **UnionType** (p. 2584) member . . . 2578

dds::core::xtypes::UnionType
<<***value-type***>> (p. *350*) Represents an IDL `union` type 2584
rti::core::UnregisterThreadOnExit
<<***extension***>> (p. *354*) Utility that calls **rti::core::unregister_thread** (p. 437) when leaving scope 2589
dds::core::UnsupportedError
Indicates that the application used an unsupported operation 2590
dds::core::policy::UserData
Attaches a buffer of opaque data that is distributed by **Built-in Topics** (p. 46) during discovery . . . 2591
rti::sub::ValidLoanedSamples< T >
<<***extension***>> (p. *354*) <<***C++11***>> (p. *353*) <<***move-only-type***>> (p. *353*) Provides access to only those samples that contain valid data 2595
rti::sub::ValidSampleIterator< T >
A forward iterator adapter that skips invalid samples 2600
dds::core::Value< D > 2601
dds::core::vector< T >
<<***value-type***>> (p. *350*) A vector convertible to std::vector and with similar functionality 2603
rti::core::VendorId
<<***extension***>> (p. *354*) Represents the vendor of the service implementing the RTPS protocol 2612
rti::config::Verbosity_def
The definition of the **dds::core::safe_enum** (p. 2265) Verbosity 2613
dds::sub::status::ViewState
Indicates whether or not an instance is new 2614
dds::core::cond::WaitSet
<<***reference-type***>> (p. *351*) Allows an application to wait until one or more of the attached **Condition** (p. 948) objects have a trigger_value of true or else until the timeout expires 2617
rti::core::cond::WaitSetProperty
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Specifies the **dds::core::cond::WaitSet** (p. 2617) behavior for multiple trigger events 2627
dds::core::WeakReference< T > 2630
rti::core::policy::WireProtocol
<<***extension***>> (p. *116*) Configures the write protocol of a **dds::domain::DomainParticipant** (p. 1326) 2630
rti::core::policy::WireProtocolAutoKind_def
<<***extension***>> (p. *354*) The definition of the **dds::core::safe_enum** (p. 2265) WireProtocol↩ AutoKind 2641
rti::pub::WriteParams
<<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) Advanced parameters for writing with a DataWriter 2642
rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >
<<***extension***>> (p. *354*) A class to inherit from when implementing a writer-side custom content filter 2652
rti::topic::WriterContentFilterHelper< T, CompileData, WriterFilterData >
<<***extension***>> (p. *354*) A class to inherit from when implementing a writer-side custom content filter 2657
dds::core::policy::WriterDataLifecycle
Controls how a **dds::pub::DataWriter** (p. 1138) handles the lifecycle of the instances (keys) that it writes 2660
dds::core::xtypes::WStringType
<<***value-type***>> (p. *350*) Represents an IDL `wstring` type 2665
rti::config::compliance::XTypesMask
This mask configures different aspects of the Extended CDR encoding (XCDR and XCDR2) 2666

Chapter 6

Module Documentation

6.1 Clock Selection

APIs related to clock selection.

APIs related to clock selection.

RTI Connext uses clocks to measure time and generate timestamps.

RTI Connext uses three clocks, an "internal clock," an "external clock," and a "wall clock". The internal clock measures time internally. The external clock generates timestamps, such as the source timestamp and the reception timestamp, in addition to providing the current time given by the Domain Participant's API. The internal and external clocks can be configured via properties. The wall clock is not configurable and always shows the system time. It is used internally whenever a real-world time is necessary, such as for external document time verification.

6.1.1 Available Clocks

Two clock implementations are generally available, the monotonic clock and the realtime clock.

The monotonic clock provides times that are monotonic from a clock that is not adjustable. This clock is useful to use in order to not be subject to changes in the system or realtime clock, which may be adjusted by the user or via time synchronization protocols. However, this time generally starts from an arbitrary point in time, such as system startup. Note that this clock is not available for all architectures. Please see the `Platform Notes` for the architectures on which it is supported. For the purposes of clock selection, this clock can be referenced by the name "monotonic".

The realtime clock provides the real time of the system. This clock may generally be monotonic but may not be guaranteed to be so. It is adjustable and may be subject to small and large changes in time. The time obtained from this clock is generally a meaningful time in that it is the amount of time from a known epoch. For the purposes of clock selection, this clock can be referenced by the names "realtime" or "system".

6.1.2 Clock Selection Strategy

By default, the internal clock uses the monotonic clock and the external clocks uses the realtime clock. The default setup offers robustness to changes in the system time. Since the monotonic clock is not available on all platforms, by default, RTI Connext will attempt to use the monotonic clock, and will fall back to the realtime clock if the monotonic clock is not available.

If you want your application to be robust to changes in the system time, you are not relying on source timestamps, and you want to avoid obtaining times from both clocks, you may use the monotonic clock for both the internal and external clocks.

6.1.3 Configuring Clock Selection

To configure the clock selection, use the **PROPERTY** (p. 541) QoS policy associated with the **dds::domain::Domain↩ Participant** (p. 1326).

See also

rti::core::policy::Property (p. 1969)

The following table lists the supported clock selection properties.

Table 6.1 ***Clock Selection Properties***

Property	Description
dds.clock.external_clock	Comma-delimited list of clocks to use for the external clock, in order of preference. Valid clock names are "realtime", "system", and "monotonic". Default: "realtime"
dds.clock.internal_clock	Comma-delimited list of clocks to use for the internal clock, in order of preference. Valid clock names are "realtime", "system", and "monotonic". Default: "monotonic" if available, "realtime" if not

6.2 Domain Module

Contains the **dds::domain::DomainParticipant** (p. 1326) class that acts as an entrypoint of RTI Connext and acts as a factory for many of the classes. The **dds::domain::DomainParticipant** (p. 1326) also acts as a container for the other objects that make up RTI Connext.

Modules

- **DomainParticipants**

  dds::domain::DomainParticipant *(p. 1326) entity and associated elements*
- **Built-in Topics**

  Built-in objects created by RTI Connext but accessible to the application.

6.2.1 Detailed Description

Contains the **dds::domain::DomainParticipant** (p. 1326) class that acts as an entrypoint of RTI Connext and acts as a factory for many of the classes. The **dds::domain::DomainParticipant** (p. 1326) also acts as a container for the other objects that make up RTI Connext.

6.3 DomainParticipants

dds::domain::DomainParticipant (p. 1326) entity and associated elements

Classes

- class **dds::domain::DomainParticipantListener**

  The listener class for a ***DomainParticipant*** *(p. 1326).*
- class **dds::domain::NoOpDomainParticipantListener**

  A convenience implementation of ***DomainParticipantListener*** *(p. 1382) where all methods are overridden to do nothing.*
- class **dds::domain::DomainParticipant**

  << ***reference-type*** *>> (p. 351) Container for all* ***dds::core::Entity*** *(p. 1520) objects.*
- class **rti::core::status::DomainParticipantProtocolStatus**
- class **rti::core::status::InvalidLocalAdvanceNoticeStatus**

  << ***extension*** *>> (p. 354) Information about the status* ***dds::core::status::StatusMask::invalid_local_advance_notice()*** *(p. 2388)*
- class **rti::domain::DomainParticipantConfigParams**

  << ***extension*** *>> (p. 354) <<* ***value-type*** *>> (p. 350) Input paramaters for creating a participant from xml configuration. It allows modification of some of the properties of the entities defined in the configuration.*
- class **dds::domain::qos::DomainParticipantFactoryQos**

  << ***value-type*** *>> (p. 350) Container of the QoS policies that do not apply to a specific entity*
- class **dds::domain::qos::DomainParticipantQos**

  << ***value-type*** *>> (p. 350) Container of the QoS policies that a* ***dds::domain::DomainParticipant*** *(p. 1326) supports*

6.3.1 Detailed Description

dds::domain::DomainParticipant (p. 1326) entity and associated elements

6.4 Built-in Topics

Built-in objects created by RTI Connext but accessible to the application.

Modules

- **Participant Built-in Topics**

  Builtin topic for accessing information about the DomainParticipants discovered by RTI Connext.
- **Topic Built-in Topics**
- **Publication Built-in Topics**

  Builtin topic for accessing information about the Publications discovered by RTI Connext.
- **Subscription Built-in Topics**

  Builtin topic for accessing information about the Subscriptions discovered by RTI Connext.
- **ServiceRequest Built-in Topic**

  Builtin topic for accessing requests from different services within RTI Connext.

6.4.1 Detailed Description

Built-in objects created by RTI Connext but accessible to the application.

RTI Connext must discover and keep track of the remote entities, such as new participants in the domain. This information may also be important to the application, which may want to react to this discovery, or else access it on demand.

A set of built-in topics and corresponding **dds::sub::DataReader** (p. 976) objects are introduced to be used by the application to access this discovery information.

The information can be accessed as if it were normal application data. This allows the application to know when there are any changes in those values by means of the **Listener** (p. 1637) or the **dds::core::cond::Condition** (p. 948) mechanisms.

The built-in data-readers all belong to a built-in **dds::sub::Subscriber** (p. 2410), which can be retrieved by using the method **dds::sub::builtin_subscriber** (p. 671). The built-in **dds::sub::DataReader** (p. 976) objects can be retrieved by using the operation **dds::sub::find** (p. 672), with the topic name as a parameter.

Built-in entities have default listener settings as well. The built-in **dds::sub::Subscriber** (p. 2410) and all of its built-in topics have 'nil' listeners with all statuses appearing in their listener masks (acting as a NO-OP listener that does not reset communication status). The built-in DataReaders have null listeners with no statuses in their masks.

The information that is accessible about the remote entities by means of the built-in topics includes all the QoS policies that apply to the corresponding remote Entity. These QoS policies appear as normal 'data' fields inside the data read by means of the built-in Topic. Additional information is provided to identify the Entity and facilitate the application logic.

The built-in **dds::sub::DataReader** (p. 976) will not provide data pertaining to entities created from the same **dds::domain::DomainParticipant** (p. 1326) under the assumption that such entities are already known to the application that created them.

The discovery information provided by each built-in reader won't be received until it is first looked up. And if the DomainParticipant was created in a disabled state (see **dds::core::policy::EntityFactory** (p. 1527)), the built-in subscriber and readers will need to be enabled explicitly (by the **dds::core::Entity::enable** (p. 1524) operation).

Refer to **dds::topic::ParticipantBuiltinTopicData** (p. 1908), **dds::topic::TopicBuiltinTopicData** (p. 2493), **dds::topic::SubscriptionBuiltinTopicData** (p. 2428) and **dds::topic::PublicationBuiltinTopicData** (p. 1977) for a description of all the built-in topics and their contents.

The QoS of the built-in **dds::sub::Subscriber** (p. 2410) and **dds::sub::DataReader** (p. 976) objects is given by the following table:

Table 6.2 ***QoS of built-in dds::sub::Subscriber (*** p. ***2410) and dds::sub::DataReader (*** p. ***976)***

QoS	Value
dds::core::policy::UserData (p. 2591)	0-length sequence
dds::core::policy::TopicData (p. 2500)	0-length sequence
dds::core::policy::GroupData (p. 1591)	0-length sequence
dds::core::policy::Durability (p. 1433)	**dds::core::policy::DurabilityKind_def::TRANSIENT_↩ LOCAL** (p. 1442)
dds::core::policy::DurabilityService (p. 1443)	Does not apply as **dds::core::policy::DurabilityKind_↩ def** (p. 1441) is **dds::core::policy::DurabilityKind_def::↩ TRANSIENT_LOCAL** (p. 1442)
dds::core::policy::Presentation (p. 1942)	access_scope = **dds::core::policy::PresentationAccess↩ ScopeKind_def::TOPIC** (p. 1951) coherent_access = false ordered_access = false
dds::core::policy::Deadline (p. 1255)	Period = infinite
dds::core::policy::LatencyBudget (p. 1631)	duration = 0
dds::core::policy::Ownership (p. 1899)	**dds::core::policy::OwnershipKind_def::SHARED** (p. 1906)
dds::core::policy::OwnershipStrength (p. 1906)	value = 0
dds::core::policy::Liveliness (p. 1645)	kind = **dds::core::policy::LivelinessKind_def::↩ AUTOMATIC** (p. 1655) lease_duration = 0
dds::core::policy::TimeBasedFilter (p. 2469)	minimum_separation = 0
dds::core::policy::Partition (p. 1923)	0-length sequence
dds::core::policy::Reliability (p. 2149)	kind = **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) max_blocking_time = 100 milliseconds
dds::core::policy::DestinationOrder (p. 1258)	**dds::core::policy::DestinationOrderKind_def::BY_↩ RECEPTION_TIMESTAMP** (p. 1264)
dds::core::policy::History (p. 1602)	kind = **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607) depth = 1
dds::core::policy::ResourceLimits (p. 2203)	max_samples = **dds::core::LENGTH_UNLIMITED** (p. 438) max_instances = **dds::core::LENGTH_UNLIMITED** (p. 438) max_samples_per_instance = **dds::core::LENGTH_↩ UNLIMITED** (p. 438)
dds::core::policy::ReaderDataLifecycle (p. 2137)	autopurge_nowriter_samples_delay = infinite autopurge_disposed_samples_delay = infinite
dds::core::policy::EntityFactory (p. 1527)	autoenable_created_entities = true

6.5 Topic Module

Contains the **dds::topic::Topic** (p. 2474), **dds::topic::ContentFilteredTopic** (p. 954), and MultiTopic classes, the TopicListener interface, and more generally, all that is needed by an application to define **dds::topic::Topic** (p. 2474) objects and attach QoS policies to them.

Modules

- **Topics**

  dds::topic::Topic *(p. 2474) entity and associated elements*
- **Zero Copy Transfer Over Shared Memory**

  *<<**extension**>> (p. 354) Zero Copy transfer over shared memory*
- **Built-in Types**

  RTI Connext provides a set of very simple data types for you to use with the topics in your application.
- **Built-in Topic's Trust Types**

  *Types used as part of **dds::topic::ParticipantBuiltinTopicData** (p. 1908), **dds::topic::PublicationBuiltinTopicData** (p. 1977), **dds::topic::SubscriptionBuiltinTopicData** (p. 2428) to describe Trust Plugins configuration.*
- **FlatData Topic-Types**

  *<<**extension**>> (p. 354) FlatData Language Binding for IDL topic-types*
- **DynamicType and DynamicData**

  *Describes **dds::core::xtypes::DynamicType** (p. 1504), **dds::core::xtypes::DynamicData** (p. 1461) and related types and functions.*
- **Topic traits and data-type support**

  Traits and operations associated to topic-types.
- **Custom Content Filters**

  Classes and associated types used to implement custom content filters.

6.5.1 Detailed Description

Contains the **dds::topic::Topic** (p. 2474), **dds::topic::ContentFilteredTopic** (p. 954), and MultiTopic classes, the TopicListener interface, and more generally, all that is needed by an application to define **dds::topic::Topic** (p. 2474) objects and attach QoS policies to them.

6.6 Topics

dds::topic::Topic (p. 2474) entity and associated elements

Classes

- class **dds::core::status::InconsistentTopicStatus**

  *Information about the status **dds::core::status::StatusMask::inconsistent_topic()** (p. 2379)*
- class **dds::topic::AnyTopic**

  *<<**reference-type**>> (p. 351) This class provides an non-template holder for representing a **Topic** (p. 2474) of any type*
- class **dds::topic::ContentFilteredTopic< T >**

  *<<**reference-type**>> (p. 351) Specialization of **TopicDescription** (p. 2503) that allows for content-based subscriptions.*
- class **dds::topic::Filter**

  *Defines the filter to create a **ContentFilteredTopic** (p. 954).*
- class **dds::topic::TopicInstance< T >**

  Encapsulates a sample and its associated instance handle.
- class **dds::topic::TopicListener< T >**

*The listener to notify status changes for a **dds::topic::Topic** (p. 2474).*

- class **dds::topic::NoOpTopicListener< T >**

  *A convenience implementation of **TopicListener** (p. 2508) where all methods are overridden to do nothing.*

- class **dds::topic::Topic< T >**

  *<<**reference-type**>> (p. 351) **Topic** (p. 2474) is the most basic description of the data to be published and subscribed.*

- class **dds::topic::TopicDescription< T >**

  *Abstract base class of **Topic** (p. 2474) and **ContentFilteredTopic** (p. 954).*

- class **rti::topic::ExpressionProperty**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Provides additional information about the filter expression passed to the writer_compile method of **rti::topic::WriterContentFilter** (p. 2652)*

- class **rti::topic::FilterSampleInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Provides meta information associated with the sample.*

- class **rti::topic::PrintFormatProperty**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A collection of attributes used to configure how data samples will be formatted when converted to a string.*

- class **dds::topic::qos::TopicQos**

  *<<**value-type**>> (p. 350) Container of the QoS policies that a **dds::topic::Topic** (p. 2474) supports*

Functions

- std::string **rti::topic::sql_filter_name** ()

  *<<**extension**>> (p. 354) The name of the built-in SQL filter*

- std::string **rti::topic::stringmatch_filter_name** ()

  *<<**extension**>> (p. 354) The name of the built-in StringMatch filter*

6.6.1 Detailed Description

dds::topic::Topic (p. 2474) entity and associated elements

6.6.2 Function Documentation

6.6.2.1 sql_filter_name()

```
std::string rti::topic::sql_filter_name ( )
```

*<<**extension**>> (p. 354)* The name of the built-in SQL filter

This filter can be used with a **dds::topic::ContentFilteredTopic** (p. 954) and a **rti::core::policy::MultiChannel** (p. 1749)-enabled **dds::pub::DataWriter** (p. 1138).

See also

dds::topic::Filter::name() (p. 1562)

rti::core::policy::MultiChannel::filter_name() (p. 1752)

6.6.2.2 stringmatch_filter_name()

```
std::string rti::topic::stringmatch_filter_name ( )
```

<<***extension***>> *(p. 354)* The name of the built-in StringMatch filter

This filter can be used with a **dds::topic::ContentFilteredTopic** (p. 954) and a **rti::core::policy::MultiChannel** (p. 1749)-enabled **dds::pub::DataWriter** (p. 1138).

The StringMatch Filter is a subset of the SQL filter; it only supports the MATCH relational operator on a single string field.

See also

dds::topic::Filter::name() (p. 1562)

rti::core::policy::MultiChannel::filter_name() (p. 1752)

Referenced by **rti::core::policy::LocatorFilter::filter_name()**.

6.7 Zero Copy Transfer Over Shared Memory

<<***extension***>> *(p. 354)* Zero Copy transfer over shared memory

<<***extension***>> *(p. 354)* Zero Copy transfer over shared memory

Note

For a description of Zero Copy transfer over shared memory and its benefits, and a tutorial, see the "Sending Large Data" chapter in the **RTI Connext User's Manual**.

For buildable **code examples**, see `https://community.rti.com/kb/flatdata-and-zerocopy-examples`.

Zero Copy transfer over shared memory is available in the C API, in the Traditional C++ API, and in the Modern C++ API.

6.8 Built-in Types

RTI Connext provides a set of very simple data types for you to use with the topics in your application.

Classes

- class **dds::core::BytesTopicType**

  Built-in type consisting of a variable-length array of opaque bytes.
- class **dds::core::StringTopicType**

  Built-in type consisting of a single character string.
- class **dds::core::KeyedStringTopicType**

  Built-in type consisting of a string payload and a second string that is the key.
- class **dds::core::KeyedBytesTopicType**

  Built-in type consisting of a variable-length array of opaque bytes and a string that is the key.

6.8.1 Detailed Description

RTI Connext provides a set of very simple data types for you to use with the topics in your application.

The middleware provides four built-in types:

- String: A payload consisting of a single string of characters. This type has no key.
- KeyedString: A payload consisting of a single string of characters and a second string, the key, that identifies the instance to which the sample belongs.
- Octets: A payload consisting of an opaque variable-length array of bytes. This type has no key.
- KeyedOctets: A payload consisting of an opaque variable-length array of bytes and a string, the key, that identifies the instance to which the sample belongs.

The String and KeyedString types are appropriate for simple text-based applications. The Octets and KeyedOctets types are appropriate for applications that perform their own custom data serialization, such as legacy applications still in the process of migrating to RTI Connext. In most cases, string-based or structured data is preferable to opaque data, because the latter cannot be easily visualized in tools or used with content-based filters (see **dds::topic::Content↩ FilteredTopic** (p. 954)).

The built-in types are very simple in order to get you up and running as quickly as possible. If you need a structured data type you can define your own type with exactly the fields you need in one of two ways:

- At compile time, by generating code from an IDL or XML file using the rtiddsgen utility
- At runtime, by using **dds::core::xtypes::DynamicData** (p. 1461)

6.8.2 Managing Memory for Builtin Types

When a sample is written, the DataWriter serializes it and stores the result in a buffer obtained from a pool of preallocated buffers. In the same way, when a sample is received, the DataReader deserializes it and stores the result in a sample coming from a pool of preallocated samples.

For builtin types, the maximum size of the buffers/samples and depends on the nature of the application using the builtin type.

You can configure the maximum size of the builtin types on a per-DataWriter and per-DataReader basis using the **rti::core::policy::Property** (p. 1969) in DataWriters, DataReaders or Participants.

The following table lists the supported builtin type properties to configure memory allocation. When the properties are defined in the DomainParticipant, they are applicable to all DataWriters and DataReaders belonging to the Domain↩ Participant unless they are overwrittem in the DataWriters and DataReaders.

Table 6.3 ***Builtin Types Allocation Properties***

Property	Description
dds.builtin_type.string.alloc_size	Maximum size of the strings published by the StringDataWriter or received the StringDataReader (includes the NULL-terminated character). Default: dds.builtin_type.string.max_size if defined. Otherwise, 1024.
dds.builtin_type.keyed_string.alloc_key_size	Maximum size of the keys used by the KeyedStringDataWriter or KeyedStringDataReader (includes the NULL-terminated character). Default: dds.builtin_type.keyed_string.max_key_size if defined. Otherwise, 1024.
dds.builtin_type.keyed_string.alloc_size	Maximum size of the strings published by the KeyedStringDataWriter or received by the KeyedStringDataReader (includes the NULL-terminated character). Default: dds.builtin_type.keyed_string.max_size if defined. Otherwise, 1024.
dds.builtin_type.octets.alloc_size	Maximum size of the octet sequences published the OctetsDataWriter or received by the OctetsDataReader. Default: dds.builtin_type.octets.max_size if defined. Otherwise, 2048.
dds.builtin_type.keyed_octets.alloc_key_size	Maximum size of the key published by the KeyedOctetsDataWriter or received by the KeyedOctetsDataReader (includes the NULL-terminated character). Default: dds.builtin_type.keyed_octets.max_key_size if defined. Otherwise, 1024.
dds.builtin_type.keyed_octets.alloc_size	Maximum size of the octets sequences published by a KeyedOctetsDataWriter or received by a KeyedOctetsDataReader. Default: dds.builtin_type.keyed_octets.max_size if defined. Otherwise, 2048.

The previous properties must be set consistently with respect to the corresponding $*$.max_size properties that set the maximum size of the builtin types in the typecode.

6.8.3 Typecodes for Builtin Types

The typecodes associated with the builtin types are generated from the following IDL type definitions:

```
module DDS {

    struct String {

        string value;

    };


    struct KeyedString {

        string key;

        string value;

    };
```

```
    struct Octets {
        sequence<octet> value;
    };

    struct KeyedOctets {
        string key;
        sequence<octet> value;
    };
};
```

The maximum size of the strings and sequences that will be included in the type code definitions can be configured on a per-DomainParticipant-basis by using the properties in following table.

Table 6.4 ***Properties for Allocating Size of Builtin Types, per DomainParticipant***

Property	Description
dds.builtin_type.string.max_size	Maximum size of the strings published by the StringDataWriters and received by the StringDataReaders belonging to a DomainParticipant (includes the NULL-terminated character). Default: 1024.
dds.builtin_type.keyed_string.max_key_size	Maximum size of the keys used by the KeyedStringDataWriters and KeyedStringDataReaders belonging to a DomainParticipant (includes the NULL-terminated character). Default: 1024.
dds.builtin_type.keyed_string.max_size	Maximum size of the strings published by the KeyedStringDataWriters and received by the KeyedStringDataReaders belonging to a DomainParticipant using the builtin type (includes the NULL-terminated character). Default: 1024
dds.builtin_type.octets.max_size	Maximum size of the octet sequences published by the OctetsDataWriters and received by the OctetsDataReader belonging to a DomainParticipant. Default: 2048
dds.builtin_type.keyed_octets.max_key_size	Maximum size of the keys used by the KeyedOctetsStringDataWriters and KeyedOctetsStringDataReaders belonging to a DomainParticipant (includes the NULL-terminated character). Default: 1024.
dds.builtin_type.keyed_octets.max_size	Maximum size of the octet sequences published by the KeyedOctetsDataWriters and received by the KeyedOctetsDataReaders belonging to a DomainParticipant. Default: 2048

For more information about the built-in types, including how to control memory usage and maximum lengths, please see the "Data Types and DDS Data Samples" chapter in the `User's Manual`.

6.9 Built-in Topic's Trust Types

Types used as part of **dds::topic::ParticipantBuiltinTopicData** (p. 1908), **dds::topic::PublicationBuiltinTopicData** (p. 1977), **dds::topic::SubscriptionBuiltinTopicData** (p. 2428) to describe Trust Plugins configuration.

Classes

- class **rti::topic::trust::EndpointTrustAlgorithmInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Trust Plugins algorithm information associated with the discovered endpoint.*
- class **rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Trust Plugins interception algorithm information associated with the discovered endpoint.*
- class **rti::topic::trust::EndpointTrustProtectionInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Trust Plugins Protection information associated with the discovered endpoint.*
- class **rti::topic::trust::ParticipantTrustAlgorithmInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Trust Plugins algorithm information associated with the discovered DomainParticipant.*
- class **rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Trust Plugins interception algorithm information associated with the discovered DomainParticipant.*
- class **rti::topic::trust::ParticipantTrustKeyEstablishmentAlgorithmInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.*
- class **rti::topic::trust::ParticipantTrustProtectionInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Trust Plugins Protection information associated with the discovered DomainParticipant.*
- class **rti::topic::trust::ParticipantTrustSignatureAlgorithmInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Trust Plugins signature algorithm information associated with the discovered DomainParticipant.*
- class **rti::topic::trust::TrustAlgorithmRequirements**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Type to describe Trust Plugins algorithm requirements.*

6.9.1 Detailed Description

Types used as part of **dds::topic::ParticipantBuiltinTopicData** (p. 1908), **dds::topic::PublicationBuiltinTopicData** (p. 1977), **dds::topic::SubscriptionBuiltinTopicData** (p. 2428) to describe Trust Plugins configuration.

These types are used to describe how RTPS wire serialization, discovery, and liveliness interact with the loaded Trust Plugins.

The meaning of the contents of these types may vary depending on what Trust Plugins the DomainParticipant is using. For information about how these types interact with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

6.10 Publication Module

Contains the **rti::pub::FlowController** (p. 1572), **dds::pub::Publisher** (p. 1993), and **dds::pub::DataWriter** (p. 1138) classes as well as the PublisherListener and **dds::pub::DataWriterListener** (p. 1202) interfaces, and more generally, all that is needed on the publication side.

Modules

- **Publishers**

  dds::pub::Publisher *(p. 1993) entity and associated elements*
- **Data Writers**

  dds::pub::DataWriter *(p. 1138) entity and associated elements*
- **Flow Controllers**

  *<<**extension**>> (p. 354)* ***rti::pub::FlowController*** *(p. 1572) and associated elements*
- **Multi-channel DataWriters**

  APIs related to Multi-channel DataWriters.

6.10.1 Detailed Description

Contains the **rti::pub::FlowController** (p. 1572), **dds::pub::Publisher** (p. 1993), and **dds::pub::DataWriter** (p. 1138) classes as well as the PublisherListener and **dds::pub::DataWriterListener** (p. 1202) interfaces, and more generally, all that is needed on the publication side.

"DCPS Publication package"

6.11 Publishers

dds::pub::Publisher (p. 1993) entity and associated elements

Classes

- class **dds::pub::PublisherListener**

  The listener to notify status changes for a ***dds::pub::Publisher*** *(p. 1993).*
- class **dds::pub::NoOpPublisherListener**

  A convenience implementation of ***PublisherListener*** *(p. 2006) where all methods are overridden to do nothing.*
- class **dds::pub::CoherentSet**

  *<<**value-type**>> (p. 350) A publishing application can request that a set of DDS data-sample changes be propagated in such a way that they are interpreted at the receivers' side as a cohesive set of modifications.*
- class **dds::pub::Publisher**

  *<<**reference-type**>> (p. 351) A publisher is the object responsible for the actual dissemination of publications.*
- class **dds::pub::SuspendedPublication**

  *<<**value-type**>> (p. 350) Indicates that the application is about to make multiple modifications using several* ***dds::pub::DataWriter*** *(p. 1138)'s belonging to the same* ***dds::pub::Publisher*** *(p. 1993)*
- class **dds::pub::qos::PublisherQos**

  *<<**value-type**>> (p. 350) Container of the QoS policies that a* ***dds::pub::Publisher*** *(p. 1993) supports*

6.11.1 Detailed Description

dds::pub::Publisher (p. 1993) entity and associated elements

6.12 Data Writers

dds::pub::DataWriter (p. 1138) entity and associated elements

Classes

- class **dds::core::status::LivelinessLostStatus**

  *Information about the status **dds::core::status::StatusMask::liveliness_lost()** (p. 2383)*
- class **dds::core::status::OfferedDeadlineMissedStatus**

  *Information about the status **dds::core::status::StatusMask::offered_deadline_missed()** (p. 2379)*
- class **dds::core::status::OfferedIncompatibleQosStatus**

  *Information about the status **dds::core::status::StatusMask::offered_incompatible_qos()** (p. 2380)*
- class **dds::core::status::PublicationMatchedStatus**

  *Information about the status **dds::core::status::StatusMask::publication_matched()** (p. 2384)*
- class **dds::pub::AnyDataWriter**

  *<<**reference-type**>> (p. 351) This class provides an non-template holder for representing a **DataWriter** (p. 1138) of any type*
- class **dds::pub::AnyDataWriterListener**

  *The listener to notify status changes for a **dds::pub::DataWriter** (p. 1138) of a generic type.*
- class **dds::pub::DataWriterListener< T >**

  *The **Listener** (p. 1637) to notify status changes for a **dds::pub::DataWriter** (p. 1138).*
- class **dds::pub::NoOpDataWriterListener< T >**

  *A convenience implementation of **DataWriterListener** (p. 1202) where all methods are overridden to do nothing.*
- class **dds::pub::DataWriter< T >**

  *<<**reference-type**>> (p. 351) Allows an application to publish data for a **dds::topic::Topic** (p. 2474)*
- class **rti::core::status::DataWriterCacheStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::datawriter_cache()** (p. 2386)*
- class **rti::core::status::ReliableWriterCacheChangedStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385)*
- class **rti::core::status::ReliableReaderActivityChangedStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::reliable_reader_activity_changed()** (p. 2385)*
- class **rti::core::status::DataWriterProtocolStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::datawriter_protocol()** (p. 2386)*
- class **rti::core::status::ServiceRequestAcceptedStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::service_request_accepted()** (p. 2388)*
- class **rti::pub::AcknowledgmentInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Information about an application-acknowledged sample*
- class **dds::pub::qos::DataWriterQos**

  *<<**value-type**>> (p. 350) Container of the QoS policies that a **dds::pub::DataWriter** (p. 1138) supports*
- class **rti::pub::WriteParams**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Advanced parameters for writing with a DataWriter*

6.12.1 Detailed Description

dds::pub::DataWriter (p. 1138) entity and associated elements

6.13 Flow Controllers

<<***extension***>> *(p. 354)* **rti::pub::FlowController** (p. 1572) and associated elements

Classes

- struct **rti::pub::FlowControllerSchedulingPolicy_def**

  <<***extension***>> *(p. 354) Kinds of flow controller shceduling policy*
- class **rti::pub::FlowControllerTokenBucketProperty**

  <<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350) Configures a* ***FlowController*** *(p. 1572) based on a tocken-bucket mechanism*
- class **rti::pub::FlowControllerProperty**

  <<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350) Configures a* ***FlowController*** *(p. 1572)*
- class **rti::pub::FlowController**

  <<***extension***>> *(p. 354)* <<***reference-type***>> *(p. 351) A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous* ***dds::pub::DataWriter*** *(p. 1138) instances are allowed to write data.*

Typedefs

- typedef **dds::core::safe_enum**< **FlowControllerSchedulingPolicy_def** > **rti::pub::FlowController**↩ **SchedulingPolicy**

  <<***extension***>> *(p. 354) The safe enumeration for* ***FlowControllerSchedulingPolicy_def::type*** *(p. 1581)*

Variables

- static OMG_DDS_API_CLASS_VARIABLE const std::string **rti::pub::FlowController::DEFAULT_NAME**

  Name that identifies the built-in default ***FlowController*** *(p. 1572).*
- static OMG_DDS_API_CLASS_VARIABLE const std::string **rti::pub::FlowController::FIXED_RATE_NAME**

  Name that identifies the built-in fixed-rate ***FlowController*** *(p. 1572).*
- static OMG_DDS_API_CLASS_VARIABLE const std::string **rti::pub::FlowController::ON_DEMAND_NAME**

  Name that identifies the built-in on-demand ***FlowController*** *(p. 1572).*

6.13.1 Detailed Description

<<***extension***>> *(p. 354)* **rti::pub::FlowController** (p. 1572) and associated elements

rti::pub::FlowController (p. 1572) provides the network traffic shaping capability to asynchronous **dds::pub::DataWriter** (p. 1138) instances. For use cases and advantages of publishing asnychronously, please refer to **rti::core::policy::PublishMode** (p. 2013) of **dds::pub::qos::DataWriterQos** (p. 1226).

See also

rti::core::policy::PublishMode (p. 2013)

rti::core::policy::PublishMode (p. 2013)

rti::core::policy::AsynchronousPublisher (p. 832)

6.13.2 Typedef Documentation

6.13.2.1 FlowControllerSchedulingPolicy

```
typedef dds::core::safe_enum< FlowControllerSchedulingPolicy_def> rti::pub::FlowController
SchedulingPolicy
```

<<***extension***>> *(p. 354)* The safe enumeration for **FlowControllerSchedulingPolicy_def::type** (p. 1581)

See also

FlowControllerSchedulingPolicy_def (p. 1581)

6.13.3 Variable Documentation

6.13.3.1 DEFAULT_NAME

```
OMG_DDS_API_CLASS_VARIABLE const std::string rti::pub::FlowController::DEFAULT_NAME [static]
```

Name that identifies the built-in default **FlowController** (p. 1572).

RTI Connext provides several built-in **rti::pub::FlowController** (p. 1572) for use with an asynchronous **dds::pub::DataWriter** (p. 1138). The user can choose to use the built-in flow controllers and optionally modify their properties or can create a custom flow controller.

By default, flow control is disabled. That is, the built-in **rti::pub::FlowController::DEFAULT_NAME** (p. 58) flow controller does not apply any flow control. Instead, it allows data to be sent asynchronously as soon as it is written by the **dds::pub::DataWriter** (p. 1138).

Essentially, this is equivalent to a user-created **rti::pub::FlowController** (p. 1572) with the following **rti::pub::FlowControllerProperty** (p. 1577) settings:

- **rti::pub::FlowControllerProperty::scheduling_policy** (p. 1579) = **FlowControllerSchedulingPolicy_def::↩ EARLIEST_DEADLINE_FIRST** (p. 1582)

- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `max_tokens` = **dds::core::LENGTH_UNLIMITED** (p. 438)

- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `tokens_added_per_period` = **dds::core::↩ LENGTH_UNLIMITED** (p. 438)

- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `tokens_leaked_per_period` = 0

- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `period` = 60 seconds

- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `bytes_per_token` = **dds::core::LENGTH_↩ UNLIMITED** (p. 438)

See also

dds::pub::DataWriter (p. 1138) constructors

rti::pub::find_flow_controller (p. 750)

rti::pub::FlowController::property (p. 1574)

rti::core::policy::PublishMode (p. 2013)

rti::core::policy::AsynchronousPublisher (p. 832)

6.13.3.2 FIXED_RATE_NAME

```
OMG_DDS_API_CLASS_VARIABLE const std::string rti::pub::FlowController::FIXED_RATE_NAME  [static]
```

Name that identifies the built-in fixed-rate **FlowController** (p. 1572).

RTI Connext provides several builtin **rti::pub::FlowController** (p. 1572) for use with an asynchronous **dds::pub::↩ DataWriter** (p. 1138). The user can choose to use the built-in flow controllers and optionally modify their properties or can create a custom flow controller.

The built-in **rti::pub::FlowController::FIXED_RATE_NAME** (p. 59) flow controller shapes the network traffic by allowing data to be sent only once every second. Any accumulated samples destined for the same destination are coalesced into as few network packets as possible.

Essentially, this is equivalent to a user-created **rti::pub::FlowController** (p. 1572) with the following **rti::pub::Flow↩ ControllerProperty** (p. 1577) settings:

- **rti::pub::FlowControllerProperty::scheduling_policy** (p. 1579) = **FlowControllerSchedulingPolicy_def::↩ EARLIEST_DEADLINE_FIRST** (p. 1582)

- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `max_tokens` = **dds::core::LENGTH_UNLIMITED** (p. 438)

- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `tokens_added_per_period` = **dds::core::LENGTH_UNLIMITED** (p. 438)

- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `tokens_leaked_per_period` = **dds::core::LENGTH_UNLIMITED** (p. 438)

- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `period` = 1 second

- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `bytes_per_token` = **dds::core::LENGTH_UNLIMITED** (p. 438)

See also

dds::pub::DataWriter (p. 1138) constructors

rti::pub::find_flow_controller (p. 750)

rti::pub::FlowController::property (p. 1574)

rti::core::policy::PublishMode (p. 2013)

rti::core::policy::AsynchronousPublisher (p. 832)

6.13.3.3 ON_DEMAND_NAME

```
OMG_DDS_API_CLASS_VARIABLE const std::string rti::pub::FlowController::ON_DEMAND_NAME  [static]
```

Name that identifies the built-in on-demand **FlowController** (p. 1572).

RTI Connext provides several builtin **rti::pub::FlowController** (p. 1572) for use with an asynchronous **dds::pub::DataWriter** (p. 1138). The user can choose to use the built-in flow controllers and optionally modify their properties or can create a custom flow controller.

The built-in **rti::pub::FlowController::ON_DEMAND_NAME** (p. 60) allows data to be sent only when the user calls **rti::pub::FlowController::trigger_flow** (p. 1575). With each trigger, all accumulated data since the previous trigger is sent (across all **dds::pub::Publisher** (p. 1993) or **dds::pub::DataWriter** (p. 1138) instances). In other words, the network traffic shape is fully controlled by the user. Any accumulated samples destined for the same destination are coalesced into as few network packets as possible.

This external trigger source is ideal for users who want to implement some form of closed-loop flow control or who want to only put data on the wire every so many samples (e.g. with the number of samples based on **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789)).

Essentially, this is equivalent to a user-created **rti::pub::FlowController** (p. 1572) with the following **rti::pub::FlowControllerProperty** (p. 1577) settings:

- **rti::pub::FlowControllerProperty::scheduling_policy** (p. 1579) = **FlowControllerSchedulingPolicy_def::↩ EARLIEST_DEADLINE_FIRST** (p. 1582)
- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `max_tokens` = **dds::core::LENGTH_UNLIMITED** (p. 438)
- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `tokens_added_per_period` = **dds::core::↩ LENGTH_UNLIMITED** (p. 438)
- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `tokens_leaked_per_period` = **dds::core↩ ::LENGTH_UNLIMITED** (p. 438)
- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `period` = **dds::core::Duration::infinite()** (p. 1451)
- **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) `bytes_per_token` = **dds::core::LENGTH_↩ UNLIMITED** (p. 438)

See also

dds::pub::DataWriter (p. 1138) constructors

rti::pub::find_flow_controller (p. 750)

rti::pub::FlowController::trigger_flow (p. 1575)

rti::pub::FlowController::property (p. 1574)

rti::core::policy::PublishMode (p. 2013)

rti::core::policy::AsynchronousPublisher (p. 832)

6.14 Subscription Module

Contains the **dds::sub::Subscriber** (p. 2410), **dds::sub::DataReader** (p. 976), **dds::sub::cond::ReadCondition** (p. 2133), **dds::sub::cond::QueryCondition** (p. 2059), and **rti::sub::TopicQuery** (p. 2516) classes, as well as the **dds::sub::SubscriberListener** (p. 2422) and **dds::sub::DataReaderListener** (p. 1055) interfaces, and more generally, all that is needed on the subscription side.

Modules

- **Subscribers**

  dds::sub::Subscriber *(p. 2410) entity and associated elements*
- **DataReaders**

  dds::sub::DataReader *(p. 976) entity and associated elements*
- **Data Samples**

  dds::sub::SampleInfo *(p. 2285),* ***dds::sub::status::SampleState*** *(p. 2316),* ***dds::sub::status::ViewState*** *(p. 2614),* ***dds::sub::status::InstanceState*** *(p. 1615) and associated elements*
- **SampleProcessor**

  *<<**experimental**>> (p. 355) <<**extension**>> (p. 354) Utility to concurrently read and process the data samples received by* ***dds::sub::DataReader*** *(p. 976).*

6.14.1 Detailed Description

Contains the **dds::sub::Subscriber** (p. 2410), **dds::sub::DataReader** (p. 976), **dds::sub::cond::ReadCondition** (p. 2133), **dds::sub::cond::QueryCondition** (p. 2059), and **rti::sub::TopicQuery** (p. 2516) classes, as well as the **dds::sub::SubscriberListener** (p. 2422) and **dds::sub::DataReaderListener** (p. 1055) interfaces, and more generally, all that is needed on the subscription side.

"DCPS Subscription package"

6.14.2 Access to data samples

Data is made available to the application by the following operations on **dds::sub::DataReader** (p. 976) objects: **dds::sub::DataReader::read** (p. 996), **dds::sub::DataReader::take** (p. 997), **dds::sub::DataReader::select** (p. 1003). and the other variants of **read()** (p. 1024) and **take()** (p. 1024).

The general semantics of the **`read()`** (p. `1024`) operation is that the application only gets access to the corresponding data (i.e. a precise instance value); the data remains the responsibility of RTI Connext and can be read again.

The semantics of the **`take()`** (p. `1024`) operations is that the application takes full responsibility for the data; that data will no longer be available locally to RTI Connext. Consequently, it is possible to access the same information multiple times only if all previous accesses were **read()** (p. 1024) operations, not **take()** (p. 1024).

The `select()` operation allows specifying which samples to read or take depending on several parameters.

Each of these operations returns a collection of `Data` values and associated **dds::sub::SampleInfo** (p. 2285) objects. Each data value represents an atom of data information (i.e., a value for one instance). This collection may contain samples related to the same or different instances (identified by the key). Multiple samples can refer to the same instance if the settings of the **HISTORY** (p. 533) QoS allow for it.

These operations reset the read communication statuses; see **Changes in read communication status** (p. 432).

See also

Interpretation of the SampleInfo (p. 2286)

6.14.2.1 Data access patterns

Applications access data using the following **dds::sub::DataReader** (p. 976) operations and their variants: **dds::sub::DataReader::read()** (p. 996), **dds::sub::DataReader::take()** (p. 997), and **dds::sub::DataReader::select()** (p. 1003).

These operations return a collection, **dds::sub::LoanedSamples** (p. 1663), consisting of which contain the actual data and **dds::sub::SampleInfo** (p. 2285) objects.

The way RTI Connext builds the collection depends on QoS policies set on the **dds::sub::DataReader** (p. 976) and **dds::sub::Subscriber** (p. 2410), the `source_timestamp` of the samples, and, when using **dds::sub::DataReader::select()** (p. 1003), the parameters used to build the dds::sub::Selector.

These operations are non-blocking and just deliver what is currently available.

Once the data samples are available to the DataReader, the application can read or take them.

To access data coherently, or in order, the **PRESENTATION** (p. 540) QoS must be set properly.

6.15 Subscribers

dds::sub::Subscriber (p. 2410) entity and associated elements

Classes

- class **dds::sub::SubscriberListener**

  *The listener to notify status changes for a **dds::sub::Subscriber** (p. 2410).*
- class **dds::sub::NoOpSubscriberListener**

  *A convenience implementation of **SubscriberListener** (p. 2422) where all methods are overridden to do nothing.*
- class **dds::sub::CoherentAccess**

  *<<**value-type**>> (p. 350) Controls whether RTI Connext will preserve the groupings of changes made by the publishing application by means of begin_coherent_changes and end_coherent_changes.*
- class **dds::sub::Subscriber**

  *<<**reference-type**>> (p. 351) A subscriber is the object responsible for actually receiving data from a subscription.*
- class **dds::sub::qos::SubscriberQos**

  *<<**value-type**>> (p. 350) Container of the QoS policies that a **dds::sub::Subscriber** (p. 2410) supports*

6.15.1 Detailed Description

dds::sub::Subscriber (p. 2410) entity and associated elements

6.16 DataReaders

dds::sub::DataReader (p. 976) entity and associated elements

Modules

- **Read Conditions**

  ***dds::sub::cond::ReadCondition** (p. 2133) and associated elements*
- **Query Conditions**

  ***dds::sub::cond::QueryCondition** (p. 2059) and associated elements*
- **Topic Queries**

  ***rti::sub::TopicQuery** (p. 2516) and associated elements.*

Classes

- class **dds::core::status::SampleRejectedState**

  Reasons why a sample was rejected.
- class **dds::core::status::SampleLostStatus**

  *Information about the status **dds::core::status::StatusMask::sample_lost()** (p. 2381)*
- class **dds::core::status::SampleRejectedStatus**

  *Information about the status **dds::core::status::StatusMask::sample_rejected()** (p. 2382)*
- class **dds::core::status::LivelinessChangedStatus**

  *Information about the status **dds::core::status::StatusMask::liveliness_changed()** (p. 2383)*
- class **dds::core::status::RequestedDeadlineMissedStatus**

  *Information about the status **dds::core::status::StatusMask::requested_deadline_missed()** (p. 2380)*
- class **dds::core::status::RequestedIncompatibleQosStatus**

  *Information about the status **dds::core::status::StatusMask::requested_incompatible_qos()** (p. 2381)*
- class **dds::core::status::SubscriptionMatchedStatus**

  *Information about the status **dds::core::status::StatusMask::subscription_matched()** (p. 2384)*
- class **dds::sub::AnyDataReader**

  *<<**reference-type**>> (p. 351) This class provides an non-template holder for representing a **DataReader** (p. 976) of any type*
- class **dds::sub::AnyDataReaderListener**

  *The listener to notify status changes for a **dds::sub::DataReader** (p. 976) of a generic type.*
- class **dds::sub::DataReaderListener< T >**

  *The **Listener** (p. 1637) to notify status changes for a **dds::sub::DataReader** (p. 976).*
- class **dds::sub::NoOpDataReaderListener< T >**

  *A convenience implementation of **DataReaderListener** (p. 1055) where all methods are overridden to do nothing.*
- class **dds::sub::DataReader< T >**

  *<<**reference-type**>> (p. 351) Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **dds::sub::Subscriber** (p. 2410).*
- class **dds::sub::DataReader< T >::Selector**

  *The **Selector** (p. 2320) class is used by the **DataReader** (p. 976) to compose read and take operations.*
- class **dds::sub::DataReader< T >::ManipulatorSelector**

  *A **Selector** (p. 2320) class enabling the streaming API.*
- class **rti::core::status::SampleLostState**

  *<<**extension**>> (p. 354) Reasons why a sample was lost*
- class **rti::core::status::DataReaderCacheStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::datareader_cache()** (p. 2386)*
- class **rti::core::status::DataReaderProtocolStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::datareader_protocol()** (p. 2387)*
- class **dds::sub::qos::DataReaderQos**

  *<<**value-type**>> (p. 350) Container of the QoS policies that a **dds::sub::DataReader** (p. 976) supports*

6.16.1 Detailed Description

dds::sub::DataReader (p. 976) entity and associated elements

6.17 Read Conditions

dds::sub::cond::ReadCondition (p. 2133) and associated elements

Classes

- class **dds::sub::cond::ReadCondition**

  <<***reference-type***>> (p. *351) Condition specifically dedicated to read operations and attached to one* ***dds::sub::DataReader*** *(p. 976).*

6.17.1 Detailed Description

dds::sub::cond::ReadCondition (p. 2133) and associated elements

6.18 Query Conditions

dds::sub::cond::QueryCondition (p. 2059) and associated elements

Classes

- class **dds::sub::cond::QueryCondition**

  <<***reference-type***>> (p. *351) Specialized* ***ReadCondition*** *(p. 2133) that allows applications to also specify a filter on the data available in a* ***dds::sub::DataReader*** *(p. 976)*
- class **dds::sub::Query**

  <<***value-type***>> (p. *350) Encapsulates a query for a* ***dds::sub::cond::QueryCondition*** *(p. 2059).*

Functions

- dds::sub::cond::detail::QueryCondition **rti::sub::cond::create_query_condition_ex** (const **dds::sub::Query** &query, const **::rti::sub::status::DataStateEx** &status)

  <<***extension***>> (p. *354) Creates a QueryCondition with the extended DataStateEx.*
- template<typename Functor >
  dds::sub::cond::detail::QueryCondition **rti::sub::cond::create_query_condition_ex** (const **dds::sub::Query** &query, const **::rti::sub::status::DataStateEx** &status, const Functor &handler)

  <<***extension***>> (p. *354) Creates a QueryCondition with the extended DataStateEx and a handler.*

6.18.1 Detailed Description

dds::sub::cond::QueryCondition (p. 2059) and associated elements

6.18.2 Function Documentation

6.18.2.1 create_query_condition_ex() [1/2]

```
dds::sub::cond::detail::QueryCondition rti::sub::cond::create_query_condition_ex (
            const  dds::sub::Query & query,
            const  ::rti::sub::status::DataStateEx & status )  [inline]
```

<<***extension***>> *(p. 354)* Creates a QueryCondition with the extended DataStateEx.

Note

This is a standalone function in the namespace rti::sub::cond

The usual way to create a **dds::sub::cond::QueryCondition** (p. 2059) is the constructor that receives a **dds::sub↩ ::status::DataState** (p. 1118). This extension function allows using **rti::sub::status::DataStateEx** (p. 1126), which includes additional state masks.

6.18.2.2 create_query_condition_ex() [2/2]

```
template<typename Functor >
dds::sub::cond::detail::QueryCondition rti::sub::cond::create_query_condition_ex (
            const  dds::sub::Query & query,
            const  ::rti::sub::status::DataStateEx & status,
            const Functor & handler )
```

<<***extension***>> *(p. 354)* Creates a QueryCondition with the extended DataStateEx and a handler.

Note

This is a standalone function in the namespace rti::sub::cond

The usual way to create a **dds::sub::cond::QueryCondition** (p. 2059) is the constructor that receives a **dds::sub↩ ::status::DataState** (p. 1118). This extension function allows using **rti::sub::status::DataStateEx** (p. 1126), which includes additional state masks.

6.19 Data Samples

dds::sub::SampleInfo (p. 2285), **dds::sub::status::SampleState** (p. 2316), **dds::sub::status::ViewState** (p. 2614), **dds::sub::status::InstanceState** (p. 1615) and associated elements

Modules

- **Sample Information**
- **Data State**

  Describes DataState, which includes SampleState, ViewState and InstanceState. .

Classes

- class **dds::sub::SharedSamples< T, DELEGATE >**

  *<<**reference-type**>> (p. 351) A sharable and container-safe version of **LoanedSamples** (p. 1663).*
- class **dds::sub::Sample< T >**

  *<<**value-type**>> (p. 350) This class encapsulate the data and meta-data associated with DDS samples.*
- class **dds::sub::SampleInfo**

  *<<**value-type**>> (p. 350) Information that accompanies each sample received by a **DataReader** (p. 976)*
- class **rti::core::CoherentSetInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A CoherentSampleInfo provides information about the coherent set associated with a sample.*
- class **rti::sub::LoanedSample< T >**

  *The element type of a **dds::sub::LoanedSamples** (p. 1663) collection.*
- class **dds::sub::LoanedSamples< T >**

  *<<**move-only-type**>> (p. 353) Provides temporary access to a collection of samples (data and info) from a **DataReader** (p. 976).*
- struct **rti::sub::IsValidData< T >**

  *<<**extension**>> (p. 354) A functor that returns true when a sample has valid data.*
- class **rti::sub::ValidLoanedSamples< T >**

  *<<**extension**>> (p. 354) <<**C++11**>> (p. 353) <<**move-only-type**>> (p. 353) Provides access to only those samples that contain valid data*
- class **rti::sub::SampleIterator< T >**

  *A random-access iterator of **LoanedSample** (p. 1659).*
- class **rti::sub::ValidSampleIterator< T >**

  A forward iterator adapter that skips invalid samples.

6.19.1 Detailed Description

dds::sub::SampleInfo (p. 2285), **dds::sub::status::SampleState** (p. 2316), **dds::sub::status::ViewState** (p. 2614), **dds::sub::status::InstanceState** (p. 1615) and associated elements

6.20 Topic Queries

rti::sub::TopicQuery (p. 2516) and associated elements.

Classes

- struct **rti::sub::TopicQuerySelectionKind_def**

  *The definition of the **dds::core::safe_enum*** (p. 2265) ***rti::sub::TopicQuerySelectionKind*** (p. 71).
- class **rti::sub::TopicQuerySelection**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Specifies the data query that defines a **TopicQuery*** (p. 2516).
- class **rti::sub::TopicQueryData**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Provides information about a **TopicQuery*** (p. 2516)
- class **rti::sub::TopicQuery**

  *<<**extension**>> (p. 354) <<**reference-type**>> (p. 351) Allows a **dds::sub::DataReader*** (p. 976) *to query the sample cache of its matching dds::pub::DataWriters.*

Typedefs

- typedef **dds::core::safe_enum**< **TopicQuerySelectionKind_def** > **rti::sub::TopicQuerySelectionKind**

  Safe Enumeration (p. 429) *of **TopicQuerySelectionKind_def*** (p. 2528)

6.20.1 Detailed Description

rti::sub::TopicQuery (p. 2516) and associated elements.

TopicQueries allow a **dds::sub::DataReader** (p. 976) to query the sample cache of its matching **dds::pub::DataWriter** (p. 1138). When the application
creates a TopicQuery, DDS will propagate it to other DomainParticipants and their DataWriters. When a DataWriter matching with the DataReader that created the TopicQuery receives it, it will send the cached samples that pass the TopicQuery's filter.

To configure how to dispatch a TopicQuery, the **dds::pub::qos::DataWriterQos** (p. 1226) includes the **rti::core::policy::TopicQueryDispatch** (p. 2522) policy. By default, a DataWriter ignores TopicQueries unless they are explicitly enabled using this policy.

The delivery of TopicQuery samples occurs in a separate RTPS channel. This allows DataReaders to receive TopicQuery samples and live samples in parallel. This is a key difference with respect to the Durability QoS policy.

Late-joining DataWriters will also discover existing TopicQueries.

After deleting a TopicQuery, new DataWriters won't discover it and existing DataWriters currently publishing cached samples may stop before delivering all of them. See **rti::sub::TopicQuery::close()** (p. 2518).

By default, a TopicQuery queries the samples that were in the DataWriter's queue at the time the DataWriter received the TopicQuery. These TopicQueries are snapshot TopicQueries. However, a TopicQuery can be created in "continuous" mode; in this case, a DataWriter will continue delivering samples that pass a continuous TopicQuery filter until the DataReader application explicitly deletes it. These TopicQueries are continuous TopicQueries.

The samples received in response to a TopicQuery are stored in the associated DataReader's cache. Any of the read/take operations can retrieve TopicQuery samples. The field **dds::sub::SampleInfo::topic_query_guid** (p. 2294) associates each sample to its TopicQuery. If the read sample is not in response to a TopicQuery then this field will be **rti::core::Guid::unknown()** (p. 1597). Note that the same data may be received several times, depending on how many TopicQueries the DataReader creates and their TopicQuerySelection.

You can choose to read or take only TopicQuery samples, only live samples, or both. To support this ReadConditions and QueryConditions provide the rti::sub::cond::create_query_condition_ex and rti::sub::cond::create_read_condition_ex APIs.

Each TopicQuery is identified by a GUID that can be accessed using the **rti::sub::TopicQuery::guid** (p. 2518) method.

6.20.2 Snapshot TopicQueries

A snapshot TopicQuery queries the samples that are in a DataWriter's queue when the DataWriter receives the TopicQuery.

When a DataWriter receives a snapshot TopicQuery, it registers the last sequence number available in the DataWriter queue (known as the cut-off SN). Then, the DataWriter applies the TopicQuery expression to all the samples up to the cut-off SN. If a sample with a SN smaller than or equal to the cut-off SN passes the filter expression, it will be included in the TopicQuery response.

When a DataWriter responds to a snapshot TopicQuery, it doesn't create any copies of the samples that are part of the response. As a result, a sample that passes the TopicQuery expression and should have been included in the response may be removed from the DataWriter queue before it reaches the DataReader that issued the TopicQuery. This can result in incomplete TopicQueries, which may or may not be necessary for your application to detect.

For instance, your application may issue a TopicQuery on a keyed topic using a filter on key-only fields and expect to receive the latest value for an instance that passes the filter. However, if the latest value for the instance is replaced with a newer value before the sample is delivered to the DataReader, it is important to detect this situation and re-issue the TopicQuery.

On the other hand, some applications may not care about which samples are returned in response to a TopicQuery, and are only interested in a snapshot of the DataWriter's queue. In such cases, the concept of incomplete TopicQueries can be ignored.

The application can detect incomplete TopicQueries by checking the **rti::core::SampleFlag::incomplete_snapshot_topic_query** (p. 2281) bit in the **dds::sub::SampleInfo::flag** (p. 2293) field of the last sample received for a snapshot TopicQuery.

To reduce the probability of receiving incomplete responses, DataWriters configured with **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607) recalculate the cut-off SN for a snapshot TopicQuery if a sample is replaced in the DataWriter queue due to KEEP_LAST replacement before the complete TopicQuery response is published.

Snapshot TopicQueries are not automatically deleted once the DataWriters that respond to the TopicQuery have finished delivering the samples included in the response. It is the responsibility of the user to delete the TopicQuery when it is no longer needed. To detect the end of a snapshot TopicQuery response, check for the **rti::core::SampleFlag::intermediate_topic_query_sample** (p. 2281) flag in the **dds::sub::SampleInfo::flag** (p. 2293) field of the samples included in the response. For more information on how to use this flag, see **Reading TopicQuery Samples** (p. 69).

6.20.3 Reading TopicQuery Samples

A TopicQuery response is the set of samples published by a DataWriter in response to a TopicQuery. Because multiple DataWriters may respond to a TopicQuery, a DataReader may receive multiple responses for a single TopicQuery.

Responses can be proxied by Routing Service DataWriters by setting <topic_query_proxy>/<mode> to PROPAGATION. These DataWriters forward a TopicQuery response, but they are not considered to be responding to the TopicQuery.

Samples that a DataReader receives in response to a TopicQuery have a value for **dds::sub::SampleInfo::topic_query_guid** (p. 2294) different than **rti::core::Guid::unknown()** (p. 1597).

In addition, the virtual GUID of the DataWriter(s) that responded to a TopicQuery is set in **dds::sub::SampleInfo::source_guid** (p. 2293).

Intermediate samples, not the final sample, have the flag **rti::core::SampleFlag::intermediate_topic_query_sample** (p. 2281) set in **dds::sub::SampleInfo::flag** (p. 2293) for a given response to a snapshot TopicQuery.

If two DataWriters respond to a snapshot TopicQuery, you should expect two final samples, one per unique **dds::sub↩ ::SampleInfo::source_guid** (p. 2293) responding to the TopicQuery, assuming that at least one sample passes the TopicQuery filter for each DataWriter.

The **dds::sub::SampleInfo::flag** (p. 2293) field of the last sample received for a snapshot TopicQuery may have the **rti::core::SampleFlag::incomplete_snapshot_topic_query** (p. 2281) bit set if the TopicQuery response is incomplete, which may or may not be necessary for your application to detect. See **Snapshot TopicQueries** (p. 69) for more information.

6.20.4 Debugging Topic Queries

There are a number of ways in which to gain more insight into what is happening in an application that is creating Topic Queries.

6.20.4.1 The Built-in ServiceRequest DataReader

TopicQueries are communicated to publishing applications through a built-in **rti::topic::ServiceRequest** (p. 2358) channel. The ServiceRequest channel is designed to be generic so that it can be used for many different purposes, one of which is TopicQueries.

When a DataReader creates a TopicQuery, a **rti::topic::ServiceRequest** (p. 2358) message is sent containing the TopicQuery information. Just as there are built-in DataReaders for **dds::topic::ParticipantBuiltinTopicData** (p. 1908), **dds::topic::SubscriptionBuiltinTopicData** (p. 2428), and **dds::topic::PublicationBuiltinTopicData** (p. 1977), there is a fourth built-in DataReader for ServiceRequests.

The new built-in DataReader can be retrieved using the built-in subscriber and **dds::sub::find** (p. 672). The topic name is **rti::topic::service_request_topic_name()** (p. 568). Installing a listener with the **dds::sub::DataReader↩ Listener::on_data_available** (p. 1058) callback implemented will allow a publishing application to be notified whenever a TopicQuery has been received from a subscribing application.

The **rti::topic::ServiceRequest::service_id** (p. 2359) of a **rti::topic::ServiceRequest** (p. 2358) corresponding to a **rti::sub::TopicQuery** (p. 2516) will be **rti::core::ServiceRequestId_def::TOPIC_QUERY** (p. 2362) and the **rti::topic↩ ::ServiceRequest::instance_id** (p. 2359) will be equal to the GUID of the **rti::sub::TopicQuery** (p. 2516) (see **rti↩ ::sub::TopicQuery::guid** (p. 2518)).

The **rti::topic::ServiceRequest::request_body** (p. 2359) is a sequence of bytes containing more information about the TopcQuery. This information can be retrieved using the **rti::sub::create_topic_query_data_from_service_↩ request()** (p. 768) function. The resulting **rti::sub::TopicQueryData** (p. 2520) contains the **rti::sub::TopicQuery↩ Selection** (p. 2526) that the **rti::sub::TopicQuery** (p. 2516) was created with, the GUID of the original DataReader that created the TopicQuery, and the topic name of that DataReader. Note: When TopicQueries are propagated through one or more Routing Services, the last DataReader that issued the TopicQuery will be a Routing Service DataReader. The **rti::sub::TopicQueryData::original_related_reader_guid** (p. 2521), however, will be that of the first DataReader to have created the TopicQuery.

6.20.4.2 The on_service_request_accepted DataWriter Listener Callback

It is possible that a **rti::topic::ServiceRequest** (p. 2358) for a **rti::sub::TopicQuery** (p. 2516) is received but is not immediately dispatched to a DataWriter. This can happen, for example, if a DataWriter was not matching with a DataReader at the time that the TopicQuery was received by the publishing application. The **dds::pub::DataWriter↩ Listener::on_service_request_accepted** (p. 1209) callback notifies a DataWriter when a ServiceRequest has been dispatched to that DataWriter. The **rti::core::status::ServiceRequestAcceptedStatus** (p. 2360) provides information about how many ServiceRequests have been accepted by the DataWriter since the last time that the status was read. The status also includes the **rti::core::status::ServiceRequestAcceptedStatus::last_request_handle** (p. 2361), which is the **dds::core::InstanceHandle** (p. 1612) of the last ServiceRequest that was accepted. This instance handle can be used to read samples per instance from the built-in ServiceRequest DataReader and correlate which ServiceRequests have been dispatched to which DataWriters.

6.20.5 Typedef Documentation

6.20.5.1 TopicQuerySelectionKind

`typedef` **`dds::core::safe_enum< TopicQuerySelectionKind_def> rti::sub::TopicQuerySelectionKind`**

Safe Enumeration (p. 429) of **TopicQuerySelectionKind_def** (p. 2528)

See also

TopicQuerySelectionKind_def (p. 2528) for the enumerated values

6.21 Sample Information

Classes

- class **dds::sub::GenerationCount**
  *<<**value-type**>> (p. 350)*
- class **dds::sub::Rank**
  *<<**value-type**>> (p. 350) Contains the sample and generation ranks of a data-sample*
- class **dds::sub::SampleInfo**
  *<<**value-type**>> (p. 350) Information that accompanies each sample received by a **DataReader** (p. 976)*

6.21.1 Detailed Description

6.22 Data State

Describes DataState, which includes SampleState, ViewState and InstanceState. .

Classes

- class **dds::sub::status::SampleState**

  Indicates whether or not a sample has ever been read.
- class **dds::sub::status::ViewState**

  Indicates whether or not an instance is new.
- class **dds::sub::status::InstanceState**

  *Indicates if the samples are from a live **dds::pub::DataWriter** (p. 1138) or not.*
- class **dds::sub::status::DataState**

  *The **DataState** (p. 1118) class describes the state of a sample and includes the information about the sample's **InstanceState** (p. 1615), **ViewState** (p. 2614), and **SampleState** (p. 2316).*
- class **rti::sub::status::StreamKind**

  *<<**extension**>> (p. 354) Indicates which stream to read from: live stream, topic-query stream or both*
- class **rti::sub::status::DataStateEx**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) An extended version of **dds::sub::status::DataState** (p. 1118) that also contains **StreamKind** (p. 2390)*

6.22.1 Detailed Description

Describes DataState, which includes SampleState, ViewState and InstanceState. .

6.23 Infrastructure Module

Defines the abstract classes and the interfaces that are refined by the other modules. Contains common definitions such as return codes, status values, and QoS policies.

Modules

- **Clock Selection**

  APIs related to clock selection.
- **Conditions and WaitSets**

  ***dds::core::cond::Condition** (p. 948) and **dds::core::cond::WaitSet** (p. 2617).*
- **Time Support**

  Time and duration types.
- **Exceptions**
- **Safe Enumeration**

  *Describes the **safe_enum** (p. 2265) class.*
- **Status Kinds**

  Kinds of communication status.
- **Supporting Types and Constants**

  Miscellaneous, general-purpose types and constants.
- **Builtin Qos Profiles**

  *<<**extension**>> (p. 354) QoS libraries and profiles that are automatically built into RTI Connext.*
- **QoS**

  Quality of Service (QoS) policies.

Classes

- class **dds::core::Entity**

  *<<**reference-type**>> (p. 351) This is the abstract base class for all the DDS objects that support QoS policies, a listener and a status condition.*
- class **rti::core::Cookie**

  Unique identifier for a written data sample in the form of a sequence of bytes.
- class **rti::core::Guid**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Class for GUID (Global Unique Identifier) representation*
- class **rti::core::QosPrintFormat**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A collection of attributes used to configure how QoS will be formatted when converted to strings.*
- class **rti::core::SampleFlag**

  *<<**extension**>> (p. 354) A set of flags that can be associated with a sample.*
- class **rti::core::SampleIdentity**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A **SampleIdentity** (p. 2282) defines a pair (Virtual Writer **Guid** (p. 1596), **SequenceNumber** (p. 2335)) that uniquely identifies a sample within a DDS domain and a Topic.*
- class **rti::core::SequenceNumber**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A class representing the DDS 64-bit Sequence Number*

6.23.1 Detailed Description

Defines the abstract classes and the interfaces that are refined by the other modules. Contains common definitions such as return codes, status values, and QoS policies.

"DCPS Infrastructure package"

See also

dds::core::QosProvider (p. 2025)

6.24 Multi-channel DataWriters

APIs related to Multi-channel DataWriters.

APIs related to Multi-channel DataWriters.

6.24.1 What is a Multi-channel DataWriter?

A Multi-channel **dds::pub::DataWriter** (p. 1138) is a **dds::pub::DataWriter** (p. 1138) that is configured to send data over multiple multicast addresses, according to some filtering criteria applied to the data.

To determine which multicast addresses will be used to send the data, the middleware evaluates a set of filters that are configured for the **dds::pub::DataWriter** (p. 1138). Each filter "guards" a channel (a set of multicast addresses). Each time a multi-channel **dds::pub::DataWriter** (p. 1138) writes data, the filters are applied. If a filter evaluates to true, the data is sent over that filter's associated channel (set of multicast addresses). We refer to this type of filter as a Channel Guard filter.

6.24.2 Configuration on the Writer Side

To configure a multi-channel **dds::pub::DataWriter** (p. 1138), simply define a list of all its channels in the **rti::core::policy::MultiChannel** (p. 1749).

The **rti::core::policy::MultiChannel** (p. 1749) is propagated along with discovery traffic. The value of this policy is available in **dds::topic::PublicationBuiltinTopicData::locator_filter** (p. 1988).

6.24.3 Configuration on the Reader Side

No special changes are required in a subscribing application to get data from a multichannel **dds::pub::DataWriter** (p. 1138). If you want the **dds::sub::DataReader** (p. 976) to subscribe to only a subset of the channels, use a **dds::topic::ContentFilteredTopic** (p. 954).

For more information on Multi-channel DataWriters, refer to the `User's Manual`.

6.24.4 Reliability with Multi-Channel DataWriters

6.24.4.1 Reliable Delivery

Reliable delivery is only guaranteed when the **dds::core::policy::Presentation::access_scope** (p. 1947) is set to **dds::core::policy::PresentationAccessScopeKind_def::INSTANCE** (p. 1951) and the filters in **rti::core::policy::MultiChannel** (p. 1749) are keyed-only based.

If any of the guard filters are based on non-key fields, RTI Connext only guarantees reception of the most recent data from the MultiChannel DataWriter.

6.24.4.2 Reliable Protocol Considerations

Reliability is maintained on a per-channel basis. Each channel has its own reliability channel send queue. The size of that queue is limited by **dds::core::policy::ResourceLimits::max_samples** (p. 2205) and/or **rti::core::policy::DataWriterResourceLimits::max_batches** (p. 1240).

The protocol parameters described in **rti::core::policy::DataWriterProtocol** (p. 1210) are applied per channel, with the following exceptions:

rti::core::policy::RtpsReliableWriterProtocol::low_watermark (p. 2232) and **rti::core::policy::RtpsReliableWriterProtocol::high_watermark** (p. 2233): The low watermark and high watermark control the queue levels (in number of samples) that determine when to switch between regular and fast heartbeat rates. With MultiChannel DataWriters, high_watermark and low_watermark refer to the DataWriter's queue (not the reliability channel queue). Therefore, periodic heartbeating cannot be controlled on a per-channel basis.

Important: With MultiChannel DataWriters, low_watermark and high_watermark refer to application samples even if batching is enabled. This behavior differs from the one without MultiChannel DataWriters (where low_watermark and high_watermark refer to batches).

rti::core::policy::RtpsReliableWriterProtocol::heartbeats_per_max_samples (p. 2240): This field defines the number of heartbeats per send queue. For MultiChannel DataWriters, the value is applied per channel. However, the send

queue size that is used to calculate the a piggyback heartbeat rate is defined per DataWriter (see **dds::core::policy::ResourceLimits::max_samples** (p. 2205))

Important: With MultiChannel DataWriters, heartbeats_per_max_samples refers to samples even if batching is enabled. This behavior differs from the one without MultiChannels DataWriters (where heartbeats_per_max_samples refers to batches).

With batching and MultiChannel DataWriters, the size of the DataWriter's send queue should be configured using **dds::core::policy::ResourceLimits::max_samples** (p. 2205) instead of max_batches **rti::core::policy::DataWriterResourceLimits::max_batches** (p. 1240) in order to take advantage of heartbeats_per_max_samples.

6.25 Transports

APIs related to RTI Connext pluggable transports.

Modules

- **Installing Transport Plugins**

  Installing and configuring transports used by RTI Connext.
- **Built-in Transport Plugins**

  Transport plugins delivered with RTI Connext.
- **Creating New Transport Plugins**

  Developing new transport plugins for RTI Connext.
- **Transport Plugins Configuration**

  Transport plugins configuration with RTI Connext.
- **Transport Address**

  Transport-independent addressing scheme using IPv6 presentation strings and numerically stored in network-ordered format.

6.25.1 Detailed Description

APIs related to RTI Connext pluggable transports.

6.25.2 Overview

RTI Connext has a pluggable transports architecture. The core of RTI Connext is transport agnostic; it does not make any assumptions about the actual transports used to send and receive messages. Instead, the RTI Connext core uses an abstract "transport API" to interact with the **transport plugins** which implement that API.

A transport plugin implements the abstract transport API and performs the actual work of sending and receiving messages over a physical transport. A collection of **builtin plugins** (see **Built-in Transport Plugins** (p. 83)) is delivered with RTI Connext for commonly used transports. New transport plugins can easily be created, thus enabling RTI Connext applications to run over transports that may not even be conceived yet. This is a powerful capability and that distinguishes RTI Connext from competing middleware approaches.

RTI Connext also provides a set of APIs for installing and configuring transport plugins to be used in an application. So that RTI Connext applications work out of the box, a subset of the builtin transport plugins is preconfigured by default (see **rti::core::policy::TransportBuiltin** (p. 2533)). You can "turn-off" some or all of the builtin transport plugins. In addition, you can configure other transport plugins for use by the application.

6.25.3 Transport Aliases

In order to use a transport plugin instance in an RTI Connext application, it must be registered with a **dds::domain::DomainParticipant** (p. 1326). When you register a transport, you specify a sequence of "alias" strings to symbolically refer to the transport plugin. The same alias strings can be used to register more than one transport plugin.

You can register multiple transport plugins with a **dds::domain::DomainParticipant** (p. 1326). An **alias** symbolically refers to one or more transport plugins registered with the **dds::domain::DomainParticipant** (p. 1326). Builtin transport plugin instances can be referred to using preconfigured aliases (see **TRANSPORT_BUILTIN** (p. 548)).

A transport plugin's class name is automatically used as an implicit alias. It can be used to refer to all the transport plugin instances of that class.

You can use aliases to refer to transport plugins, in order to specify:

- the transport plugins to use for **discovery** (see **rti::core::policy::Discovery::enabled_transports** (p. 1267)), and for **dds::pub::DataWriter** (p. 1138) and **dds::sub::DataReader** (p. 976) entities (see **rti::core::policy::TransportSelection** (p. 2554)).

- the **multicast** addresses on which to receive discovery messages (see **rti::core::policy::Discovery::multicast_receive_addresses** (p. 1268)), and the multicast addresses and ports on which to receive user data (see **rti::core::policy::TransportMulticast** (p. 2543)).

- the **unicast ports** used for user data (see **rti::core::policy::TransportUnicast** (p. 2556)) on both **dds::pub::DataWriter** (p. 1138) and **dds::sub::DataReader** (p. 976) entities.

- the transport plugins used to parse an address string in a locator (**Locator Format** (p. 560) and **NDDS_DISCOVERY_PEERS** (p. 559)).

A **dds::domain::DomainParticipant** (p. 1326) (and contained its entities) start using a transport plugin after the **dds::domain::DomainParticipant** (p. 1326) is enabled (see **dds::core::Entity::enable** (p. 1524)). An entity will use *all* the transport plugins that match the specified transport QoS policy. All transport plugins are treated uniformly, regardless of how they were created or registered; there is no notion of some transports being more "special" that others.

6.25.4 Transport Lifecycle

A transport plugin is owned by whomever creates it. Thus, if you create and register a transport plugin with a **dds::domain::DomainParticipant** (p. 1326), you are responsible for deleting it by calling its destructor. Note that builtin transport plugins (**TRANSPORT_BUILTIN** (p. 548)) and transport plugins that are loaded through the **PROPERTY** (p. 541) QoS policy (see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 81)) are automatically managed by RTI Connext.

A user-created transport plugin must not be deleted while it is still in use by a **dds::domain::DomainParticipant** (p. 1326). This generally means that a user-created transport plugin instance can only be deleted after the **dds::domain::DomainParticipant** (p. 1326) with which it was registered is deleted (see DomainParticipantFactory::delete_participant). Note that a transport plugin *cannot* be "unregistered" from a **dds::domain::DomainParticipant** (p. 1326).

A transport plugin instance cannot be registered with more than one **dds::domain::DomainParticipant** (p. 1326) at a time. This requirement is necessary to guarantee the multi-threaded safety of the transport API.

If the same physical transport resources are to be used with more than one **dds::domain::DomainParticipant** (p. 1326) in the same address space, the transport plugin should be written in such a way so that it can be instantiated multiple times—once for each **dds::domain::DomainParticipant** (p. 1326) in the address space. Note that it is always possible to write the transport plugin so that multiple transport plugin instances share the same underlying resources; however the burden (if any) of guaranteeing multi-threaded safety to access shared resource shifts to the transport plugin developer.

6.25.5 Transport Class Attributes

A transport plugin instance is associated with two kinds of attributes:

- the *class* attributes that are decided by the plugin writer; these are invariant across all instances of the transport plugin class, and
- the *instance* attributes that can be set on a per instance basis by the transport plugin user.

Every transport plugin must specify the following class attributes.

transport class id (see NDDS_Transport_Property_t::classid (p. **1787))** Identifies a transport plugin implementation class. It denotes a unique "class" to which the transport plugin instance belongs. The class is used to distinguish between different transport plugin implementations. Thus, a transport plugin vendor should ensure that its transport plugin implementation has a unique class.

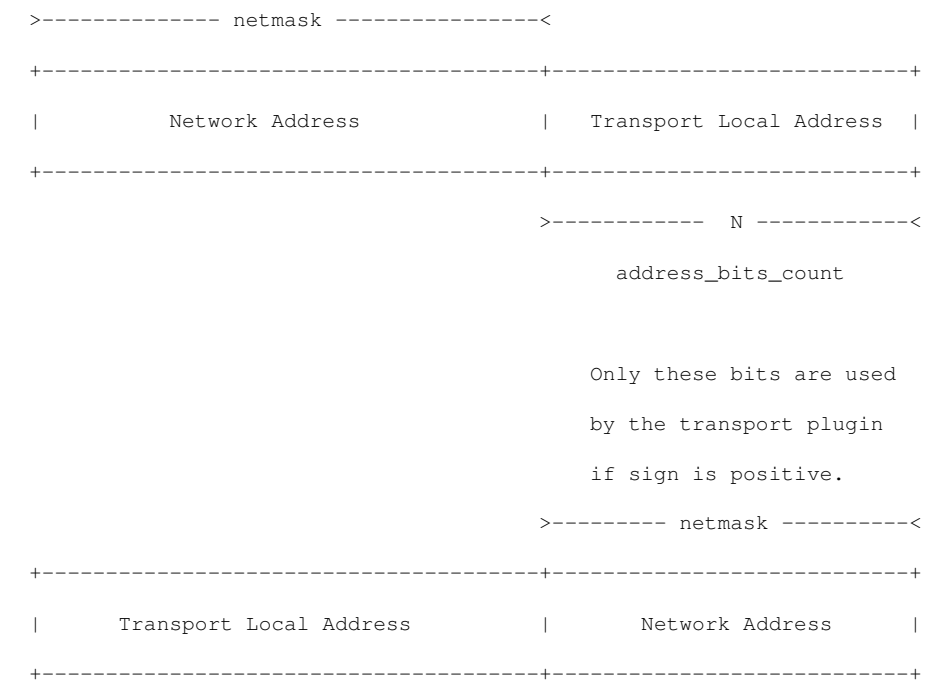
Two transport plugin instances report the same class *iff* they have compatible implementations. Transport plugin instances with mismatching classes are not allowed (by the RTI Connext Core) to communicate with one another.

Multiple implementations (possibly from different vendors) for a physical transport mechanism can co-exist in an RTI Connext application, provided they use different transport class IDs.

The class ID can also be used to distinguish between different transport protocols over the same physical transport network (e.g., UDP vs. TCP over the IP routing infrastructure).

transport significant address bit count (see NDDS_Transport_Property_t::address_bit_count (p. **1788))** RTI Connext's addressing is modeled after the IPv6 and uses 128-bit addresses (**Transport Address** (p. 372)) to route messages.

A transport plugin is expected to map the transport's internal addressing scheme to 128-bit addresses. Depending on the sign of this attribute, this mapping uses only `N` least significant bits (LSB) if positive or `N` most significant bits (MSB) if negative; these bits are specified by this attribute.



```
>-----------------  N  -----------------<

           address_bits_count


         Only these bits are used

         by the transport plugin

         if sign is negative.
```

The remaining bits of an address using the 128 - abs(bit address) representation will be considered as part of the "network address" (see **Transport Network Address** (p. 79)) and thus ignored by the transport plugin's internal addressing scheme.

For *unicast* addresses, the transport plugin is expected to ignore the higher (128 - **NDDS_Transport_Property_t::address_bit_count** (p. 1788)) bits. RTI Connext is free to manipulate those bits freely in the addresses passed in/out to the transport plugin APIs.

Theoretically, the significant address bits count, N is related to the size of the underlying transport network as follows:

$$address_bits_count >= ceil(log_2(total_addressable_transport_unicast_interfaces))$$

The equality holds when the most compact (theoretical) internal address mapping scheme is used. A practical address mapping scheme may waste some bits.

6.25.6 Transport Instance Attributes

The *per instance* attributes to configure the plugin instance are generally passed in to the plugin constructor. These are defined by the transport plugin writer, and can be used to:

- customize the behavior of an instance of a transport plugin, including the send and the receiver buffer sizes, the maximum message size, various transport level classes of service (CoS), and so on.
- specify the resource values, network interfaces to use, various transport level policies, and so on.

RTI Connext requires that every transport plugin instance must specify the **NDDS_Transport_Property_t::message_size_max** (p. 1789) and **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789).

It is up to the transport plugin developer to make these available for configuration to transport plugin user.

Note that it is important that the instance attributes are "compatible" between the sending side and the receiving side of communicating applications using different instances of a transport plugin class. For example, if one side is configured to send messages larger than can be received by the other side, then communications via the plugin may fail.

6.25.7 Transport Network Address

The address bits not used by the transport plugin for its internal addressing constitute its network address bits.

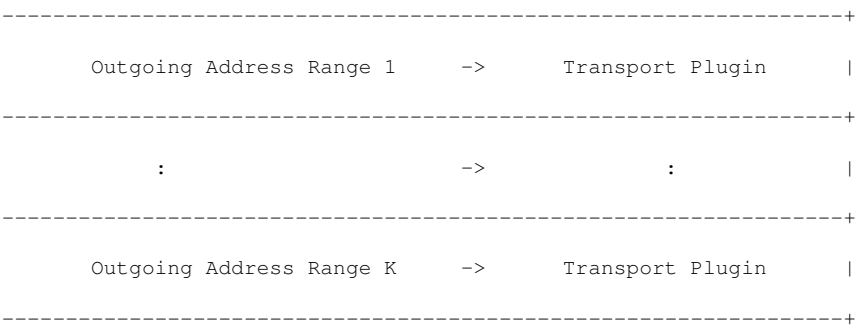
In order for RTI Connext to properly route the messages, each unicast interface in the RTI Connext *domain* must have a unique address. RTI Connext allows the user to specify the value of the network address when installing a transport plugin via the Transport_Support::register_transport() API.

The network address for a transport plugin should be chosen such that the resulting fully qualified 128-bit address will be unique in the RTI Connext domain. Thus, if two instances of a transport plugin are registered with a **dds::domain↩ ::DomainParticipant** (p. 1326), they will be at different network addresses in order for their unicast interfaces to have unique fully qualified 128-bit addresses. It is also possible to create multiple transports with the same network address, as it can be useful for certain use cases; note that this will require special entity configuration for most transports to avoid clashes in resource use (e.g. sockets for UDPv4 transport).

6.25.8 Transport Send Route

By default, a transport plugin is configured to send outgoing messages destined to addresses in the network address range at which the plugin was registered.

RTI Connext allows the user to configure the routing of outgoing messages via the Transport_Support::add_send_↩ route() API, so that a transport plugin will be used to send messages only to certain ranges of destination addresses. The method can be called multiple times for a transport plugin, with different address ranges.

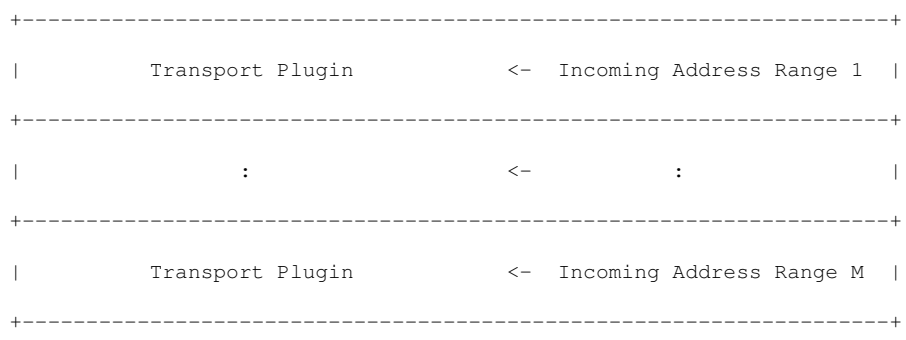


The user can set up a routing table to restrict the use of a transport plugin to send messages to selected addresses ranges.

6.25.9 Transport Receive Route

By default, a transport plugin is configured to receive incoming messages destined to addresses in the network address range at which the plugin was registered.

RTI Connext allows the user to configure the routing of incoming messages via the Transport_Support::add_receive↩ _route() API, so that a transport plugin will be used to receive messages only on certain ranges of addresses. The method can be called multiple times for a transport plugin, with different address ranges.



The user can set up a routing table to restrict the use of a transport plugin to receive messages from selected ranges. For example, the user may restrict a transport plugin to

- receive messages from a certain multicast address range.
- receive messages only on certain unicast interfaces (when multiple unicast interfaces are available on the transport plugin).

6.26 Installing Transport Plugins

Installing and configuring transports used by RTI Connext.

Installing and configuring transports used by RTI Connext.

There is more than one way to install a transport plugin for use with RTI Connext:

- If it is a builtin transport plugin, by specifying a bitmask in **rti::core::policy::TransportBuiltin** (p. 2533) (see **Built-in Transport Plugins** (p. 83))
- For all other non-builtin transport plugins, by dynamically loading the plugin through **PROPERTY** (p. 541) QoS policy settings of **dds::domain::DomainParticipant** (p. 1326) (only supported on architectures that support dynamic libraries, see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 81))

The lifecycle of the transport plugin is automatically managed by RTI Connext. See **Transport Lifecycle** (p. 76) for details.

6.26.1 Loading Transport Plugins through Property QoS Policy of Domain Participant

On architectures that support dynamic libraries, a non-builtin transport plugin written in C/C++ and built as a dynamic-link library (∗.dll/∗.so) can be loaded by RTI Connext through the **PROPERTY** (p. 541) QoS policy settings of the **dds::domain::DomainParticipant** (p. 1326).

Dynamic libraries are supported on all architectures except INTEGRITY and certain VxWorks platforms. For VxWorks, dynamic libraries are only supported for architectures that are on Pentium/Arm CPUs AND use kernel mode.

The dynamic-link library (and all the dependent libraries) need to be in the library search path during runtime (in the **LD_LIBRARY_PATH** environment variable on Linux systems, DYLD_LIBRARY_PATH on macOS systems, or **Path** on Windows systems).

To allow dynamic loading of the transport plugin, the transport plugin must implement the RTI Connext abstract transport API and must provide a function with the signature Transport_create_plugin that can be called by RTI Connext to create an instance of the transport plugin. The name of the dynamic library that contains the transport plugin implementation, the name of the function and properties that can be used to create the plugin, and the aliases and network address that are used to register the plugin can all be specified through the **PROPERTY** (p. 541) QoS policy of the **dds::domain::DomainParticipant** (p. 1326).

The following table lists the property names that are used to load the transport plugins dynamically:

Table 6.5 ***Properties for dynamically loading and registering transport plugins***

Property Name	Description	Required?
dds.transport.load_plugins	Comma-separated strings indicating the prefix names of all plugins that will be loaded by RTI Connext. Up to 8 plugins may be specified. For example, "dds.transport.TCPv4.tcp1, dds.transport.TCPv4.tcp2", In the following examples, <TRANSPORT_PREFIX> is used to indicate one element of this string that is used as a prefix in the property names for all the settings that are related to the plugin. <TRANSPORT_PREFIX> must begin with "dds.transport." (such as "dds.transport.mytransport").	**YES**
<TRANSPORT_PREFIX>.library	Should be set to the name of the dynamic library (∗.so for Linux systems, ∗.dylib for macOS systems, and ∗.dll for Windows systems) that contains the transport plugin implementation. This library (and all the other dependent dynamic libraries) needs to be in the library search path used by RTI Connext during run time (pointed to by the environment variable **LD_LIBRARY_PATH** on Linux systems, DYLD_LIBRARY_PATH on macOS systems, or **Path** on Windows systems).	**YES**
<TRANSPORT_PREFIX>.create_function	Should be set to the name of the function with the prototype of Transport_create_plugin that can be called by RTI Connext to create an instance of the plugin. The resulting transport plugin will then be registered by RTI Connext through Transport_Support::register_transport	**YES**

Property Name	Description	Required?
<TRANSPORT_PREFIX>.aliases	Used to register the transport plugin returned by Transport_create_plugin (as specified by <**TRANSPORT_PREFIX**>**.create_function**) to the **dds::domain::DomainParticipant** (p. 1326). Refer to **aliases_in** parameter in Transport_↩ Support::register_transport for details. Aliases should be specified as a comma-separated string, with each comma delimiting an alias. If it is not specified, <TRANSPORT_PREFIX> – without the leading "dds.transport" – is used as the default alias for the plugin.	NO
<TRANSPORT_PREFIX>.network_address	Used to register the transport plugin returned by Transport_create_plugin (as specified by <**TRANSPORT_PREFIX**>**.create_function**) to the **dds::domain::DomainParticipant** (p. 1326). Refer to **network_address_in** parameter in Transport_Support::register_transport for details. If it is not specified, the **network_address_out** output parameter from Transport_create_plugin is used. The default value is a zeroed out network address.	NO
<TRANSPORT_PREFIX>.<property_name>	Property that is passed into Transport_↩ create_plugin (as specified by <**TRANSPORT**↩ **_PREFIX**>**.create_function**) for creating the transport plugin. This property name-value pair will be passed to Transport_create_plugin after stripping out <TRANSPORT_PREFIX> from the property name. The parsing of this property and configuring the transport using this property should be handled by the implementation of each transport plugin. Multiple <TRANSPORT↩ _PREFIX>.<property_name> can be specified. Note: "library", "create_function", "aliases" and "network_address" cannot be used as the <property_name> due to conflicts with other builtin property names.	NO

A transport plugin is dynamically created and registered to the **dds::domain::DomainParticipant** (p. 1326) by RTI Connext when:

- the **dds::domain::DomainParticipant** (p. 1326) is enabled,
- the first DataWriter/DataReader is created, or
- you lookup a builtin DataReader (**dds::sub::find** (p. 672)),

whichever happens first.

Any changes to the transport plugin related properties in the **PROPERTY** (p. 541) QoS policy after the transport plugin has been registered with the **dds::domain::DomainParticipant** (p. 1326) will have no effect.

6.27 Built-in Transport Plugins

Transport plugins delivered with RTI Connext.

Modules

- **UDP Transport Plugin definitions**

  UDP Transport Plugin definitions.
- **Shared Memory Transport**

  Built-in transport plug-in for inter-process communications using shared memory.
- **UDPv4 Transport**

  Transport plug-in using UDP/IPv4.
- **Real-Time WAN Transport**

  Transport plug-in using UDP/IPv4 for WAN communications..
- **UDPv6 Transport**

  Transport plug-in using UDP/IPv6.

6.27.1 Detailed Description

Transport plugins delivered with RTI Connext.

The **TRANSPORT_BUILTIN** (p. 548) specifies the collection of transport plugins that can be automatically configured and managed by RTI Connext as a convenience to the user.

These transport plugins can simply be turned "on" or "off" by a specifying a bitmask in **rti::core::policy::Transport↩ Builtin** (p. 2533), thus bypassing the steps for setting up a transport plugin. RTI Connext preconfigures the transport plugin properties, the network address, and the aliases to "factory defined" values.

If a builtin transport plugin is turned "on" in **rti::core::policy::TransportBuiltin** (p. 2533), the plugin is implicitly created and registered to the corresponding **dds::domain::DomainParticipant** (p. 1326) by RTI Connext when:

- the **dds::domain::DomainParticipant** (p. 1326) is enabled,
- the first DataWriter/DataReader is created, or
- you lookup a builtin DataReader (**dds::sub::find** (p. 672)),

whichever happens first.

Each builtin transport contains its own set of properties. For example, the UDPv4Transport allows the application to specify whether or not multicast is supported, the maximum size of the message, and provides a mechanism for the application to filter out network interfaces.

The builtin transport plugin properties can be changed by the method Transport_Support::set_builtin_transport_↩ property() or by using the **PROPERTY** (p. 541) QoS policy associated with the **dds::domain::DomainParticipant** (p. 1326). Builtin transport plugin properties specified in **rti::core::policy::Property** (p. 1969) always overwrite the ones specified through Transport_Support::set_builtin_transport_property(). Refer to the specific builtin transport for the list of property names that can be specified through **PROPERTY** (p. 541) QoS policy.

Any changes to the builtin transport properties after the builtin transports have been registered with will have no effect.

See also

Transport_Support::set_builtin_transport_property() **rti::core::policy::Property** (p. 1969)

6.28 Creating New Transport Plugins

Developing new transport plugins for RTI Connext.

Developing new transport plugins for RTI Connext.

RTI Connext provides an abstract "C" language API for creating new transport plugins. If you are interested in creating a new transport plugin for RTI Connext, please contact your RTI representative or email `sales@rti.com`.

6.29 Queries and Filters Syntax

6.29.1 Syntax for DDS Queries and Filters

A subset of the WHERE clause in SQL is used in several parts of the specification:

- The `filter_expression` in the **dds::topic::ContentFilteredTopic** (p. 954)
- The `query_expression` in the **dds::sub::cond::QueryCondition** (p. 2059)
- <<***extension***>> *(p. 354)* The `filter_expression` in the **rti::sub::TopicQuerySelection** (p. 2526)
- <<***extension***>> *(p. 354)* The `filter_expression` in the **rti::core::ChannelSettings** (p. 921)

Those expressions may use a subset of SQL, extended with the possibility to use program variables in the SQL expression. The allowed SQL expressions are defined with the BNF-grammar below.

The following notational conventions are made:

- *NonTerminals* are typeset in italics.
- `'Terminals'` are quoted and typeset in a fixed width font. They are written in upper case in most cases in the BNF-grammar below, but should be case insensitive.
- **TOKENS** are typeset in bold.
- The notation (*element* // ',') represents a non-empty comma-separated list of *elements*.

6.29.2 SQL grammar in BNF

```
FilterExpression ::=  Condition
Condition   ::=  Predicate
           |    Condition  'AND'  Condition
           |    Condition  'OR'  Condition
           |    'NOT'  Condition
           |    '('  Condition  ')'
Predicate   ::=  ComparisonPredicate
           |    BetweenPredicate
ComparisonPredicate ::=  ComparisonTerm  RelOp  ComparisonTerm
ComparisonTerm      ::= FieldIdentifier
                   |  Parameter
BetweenPredicate    ::=  FieldIdentifier  'BETWEEN'  Range
                   |    FieldIdentifier  'NOT BETWEEN'  Range
FieldIdentifier      ::= FIELDNAME
                   | IDENTIFIER
RelOp     ::=  '=' |  '>' |  '>=' |  '<' |  '<=' |  '<>' |  'LIKE' |  'MATCH'
Range     ::=  Parameter  'AND'  Parameter
Parameter ::= INTEGERVALUE
         |   CHARVALUE
         |   FLOATVALUE
         |   STRING
         |   ENUMERATEDVALUE
         |   BOOLEANVALUE
         |   NULLVALUE
         |   PARAMETER
```

6.29.3 Token expression

The syntax and meaning of the tokens used in the SQL grammar is described as follows:

- **IDENTIFIER** - An identifier for a FIELDNAME, and is defined as any series of characters 'a', ..., 'z', 'A', ..., 'Z', '0', ..., '9', '_' but may not start with a digit.

Formal notation:

```
IDENTIFIER:  LETTER ( PART_LETTER)*
where  LETTER: [ "A"-"Z","_","a"-"z" ]
       PART_LETTER: [ "A"-"Z","_","a"-"z","0"-"9" ]
```

- **FIELDNAME** - A fieldname is a reference to a field in the data structure. The dot '.' is used to navigate through nested structures. The number of dots that may be used in a FIELDNAME is unlimited. The FIELDNAME can refer to fields at any depth in the data structure. The names of the field are those specified in the IDL definition of the corresponding structure, which may or may not match the fieldnames that appear on the language-specific (e.g., C/C++, Java) mapping of the structure. To reference to the *n*+1 element in an array or sequence, use the notation '[*n*]', where *n* is a natural number (zero included). FIELDNAME must resolve to a primitive IDL type; that is either boolean, octet, (unsigned) short, (unsigned) long, (unsigned) long long, float double, char, wchar, string, wstring, or enum.

Formal notation:

```
FIELDNAME:  FieldNamePart ( "."  FieldNamePart )*
where  FieldNamePart : IDENTIFIER ( "["  Index "]" )*
       Index> : (["0"-"9"])+
              | ["0x","0X"](["0"-"9", "A"-"F", "a"-"f"])+
```

Primitive IDL types referenced by FIELDNAME are treated as different types in *Predicate* according to the following table:

Predicate Data Type	IDL Type
BOOLEANVALUE	boolean
INTEGERVALUE	octet, (unsigned) short, (unsigned) long, (unsigned) long long
FLOATVALUE	float, double
CHARVALUE	char, wchar
STRING	string, wstring
ENUMERATEDVALUE	enum

- **INTEGERVALUE** - Any series of digits, optionally preceded by a plus or minus sign, representing a decimal integer value within the range of the system. 64-bit integers (int64 and uint64) must be followed by either l or L, otherwise the value is treated as a 32-bit integer. A hexadecimal number is preceded by 0x and must be a valid hexadecimal expression.

Formal notation:

```
INTEGERVALUE : (["+","-"])? (["0"-"9"])+ [("L","l")]?
             | (["+","-"])? ["0x","0X"](["0"-"9", "A"-"F", "a"-"f"])+ [("L","l")]?
```

- **CHARVALUE** - A single character enclosed between single quotes.

Formal notation:

```
CHARVALUE : "'" (~["'"])? "'"
```

- **FLOATVALUE** - Any series of digits, optionally preceded by a plus or minus sign and optionally including a floating point ('.'). A power-of-ten expression may be postfixed, which has the syntax e*n* or E*n*, where *n* is a number, optionally preceded by a plus or minus sign.

Formal notation:

```
FLOATVALUE : (["+","-"])? (["0"-"9"])* (".")? (["0"-"9"])+ ( EXPONENT)?
where  EXPONENT: ["e","E"] (["+","-"])? (["0"-"9"])+
```

- **STRING** - Any series of characters encapsulated in single quotes, except the single quote itself.

Formal notation:

```
STRING : "'" (~["'"])* "'"
```

- **ENUMERATEDVALUE** - An enumerated value is a reference to a value declared within an enumeration. Enumerated values consist of the name of the enumeration label enclosed in single quotes. The name used for the enumeration label must correspond to the label names specified in the IDL definition of the enumeration.

Formal notation:

```
ENUMERATEDVALUE : "'" ["A" - "Z", "a" - "z"] ["A" - "Z", "a" - "z", "_", "0" - "9"]* "'"
```

- **BOOLEANVALUE** - Can either be 'TRUE' or 'FALSE', case insensitive.

Formal notation (case insensitive):

```
BOOLEANVALUE : ["TRUE","FALSE"]
```

- **NULLVALUE** - Can be null, and is case insensitive.

Formal notation (case insensitive):

```
NULLVALUE : "null"
```

- **PARAMETER** - A parameter is of the form %*n*, where *n* represents a natural number (zero included) smaller than 100. It refers to the n + 1 th argument in the given context. Argument can only in primitive type value format. It cannot be a FIELDNAME.

Formal notation:

```
PARAMETER : "%" (["0"-"9"])+
```

6.29.4 String Parameters

Strings used as parameter values must contain the enclosing quotation marks (') within the parameter value, and not place the quotation marks within the expression statement. For example, the following expression is legal:

```
" symbol MATCH %0 " with parameter 0 = " 'IBM' "
```

whereas the following expression will not compile:

```
" symbol MATCH '%0' " with parameter 0 = " IBM "
```

6.29.5 Type compatability in Predicate

Only certain combination of type comparisons are valid in *Predicate*. The following table marked all the compatible pairs with 'YES':

	BOOLEANVALUE	INTEGERVALUE	FLOATVALUE	CHARVALUE	STRING	ENUMERATEDVALUE
BOOLEAN	YES					
INTEGERVALUE		YES	YES			
FLOATVALUE		YES	YES			
CHARVALUE				YES	YES	YES
STRING				YES	YES(∗1)	YES
ENUMERATEDVALUE		YES		YES(∗2)	YES(∗2)	YES(∗3)

- (∗1) See **SQL Extension: Regular Expression Matching** (p. 88)
- (∗2) Because the formal notation of the Enumeration values, they are compatible with string and char literals, but they are not compatible with string or char variables, i.e., "MyEnum='EnumValue'" would be correct, but "My↩ Enum=MyString" is not allowed.
- (∗3) Only for same type Enums.

6.29.6 SQL Extension: Regular Expression Matching

The relational operator MATCH may only be used with string fields. The right-hand operator is a string *pattern*. A string pattern specifies a template that the left-hand field value must match. The characters ,/\?∗[]-^!% have special meanings unless they are escaped by the escape character '\'.

MATCH is case-sensitive.

The pattern allows limited "wild card" matching under the following rules:

Character	Meaning
,	"," separates a list of alternate patterns. The field string is matched if it matches one or more of the patterns.
/	"/" in the pattern string matches a / in the field string. This character is used to separate a sequence of mandatory substrings.
?	"?" in the pattern string matches any single *non-special* characters in the field string.
∗	"∗" in the pattern string matches 0 or more *non-special* characters in field string.
[*charlist*]	Matches any one of the characters from the list of characters in *charlist*.
[*s-e*]	Matches any character any character from *s* to *e*, inclusive.
%	"%" is used to designate filter expressions parameters.
[!*charlist*] or [^*charlist*]	Matches any characters not in *charlist* (not supported).
[!*s-e*] or [^*s-e*]	Matches any characters not in the interval [s-e] (not supported).
\	Escape character for special characters.

The syntax is similar to the POSIX fnmatch syntax (1003.2-1992 section B.6). The MATCH syntax is also similar to the 'subject' strings of TIBCO Rendezvous.

Note: To use special characters as regular characters in regular expressions, you must escape them using the character '\'. For example, 'A[' is considered a malformed expression and the result is undefined.

6.29.7 Character Encoding

The default encoding for IDL strings is UTF-8. RTI Connext offers ISO 8859-1 as an alternative encoding for IDL strings.

In order to configure ISO 8859-1 as the encoding for filtering of IDL strings, you can set the DomainParticipant property **dds.domain_participant.filtering_character_encoding** to ISO-8859:

The possible values for **dds.domain_participant.filtering_character_encoding** are:

- **UTF-8** (default value)
- **ISO-8859-1**

6.29.8 Unicode Normalization

Unicode supports multiple ways to encode some characters, most notably accented characters. A composed character in Unicode can often have a number of different ways of representing the character. For example:

- Precomposed is represented by \u1e3c
- Composed = L + ^ is represented by \u004c + \u032d

The lexical comparison of the two characters above will return false. In order to do the correct comparison the characters need to be normalized, that is, reduced to the same character composition.

When the character encoding for filtering of IDL strings is UTF-8, the Unicode normalization behavior can be controlled using a DomainParticipant property called **dds.domain_participant.filtering_unicode_normalization**.

The possible values of the normalization property are:

- **OFF**: Disables normalization
- **NFD**: Canonical Decomposition
- **NFC (default value)**: Canonical Decomposition, followed by Canonical Composition
- **NFK**: Compatibility Decomposition, followed by Canonical Composition
- **NFKC_Casefold**: Casefold followed by NFKC normalization

Because normalization may affect performance, and it is enabled by default, the property allows disabling the normalization process per DomainParticipant using the value OFF. However, you should be aware that doing this may lead to unexpected behavior.

6.29.9 Examples

Assuming Topic "Location" has as an associated type a structure with fields "flight_id, x, y, z", and Topic "FlightPlan" has as fields "flight_id, source, destination". The following are examples of using these expressions.

Example of a **filter_expression** (for **dds::topic::ContentFilteredTopic** (p. 954)) or a **query_expression** (for **dds::sub::cond::QueryCondition** (p. 2059)):

- `"z < 1000 AND x < 23"`

Examples of a **filter_expression** using **MATCH** (for **dds::topic::ContentFilteredTopic** (p. 954)) operator:

- `"symbol MATCH 'NASDAQ/GOOG'"`
- `"symbol MATCH 'NASDAQ/[A-M]*'"`

6.30 Logging and Version

APIs of troubleshooting utilities and APIs designed to configure the overall behavior of RTI Connext.

Modules

- **Logging**

  Configure how much debugging information is reported during runtime and where it is logged.
- **Version**

  Retrieve information for the RTI Connext product, the core library, and the C, C++ or Java libraries.

6.30.1 Detailed Description

APIs of troubleshooting utilities and APIs designed to configure the overall behavior of RTI Connext.

6.31 General Utilities and Compliance Configuration

API of general utilities used in the RTI Connext distribution.

Modules

- **Compliance Configuration**

  APIs to configure compliance with certain standard specifications.
- **Discovery Snapshot**

  Utilities to print discovery snapshots of DDS entities.
- **Network Capture**

  Save network traffic into a capture file for further analysis.
- **Heap Monitoring**

  Monitor memory allocations done by the middleware on the native heap.
- **Other Utilities**

  *Other Utilities, such as **rti::util::spin()** (p. 597)*

6.31.1 Detailed Description

API of general utilities used in the RTI Connext distribution.

6.32 Observability Framework

RTI Connext Observability Framework modules and APIs.

Modules

- **Monitoring Library 2.0**

  RTI Monitoring Library 2.0 public API.

6.32.1 Detailed Description

RTI Connext Observability Framework modules and APIs.

RTI Connext Observability Framework is a holistic solution that uses telemetry data to provide deep visibility into the current and past states of your RTI Connext applications.

6.33 Request-Reply Pattern

Support for the request-reply communication pattern.

Modules

- **Requester**

  rti::request::Requester *(p. 2187) and associated elements*
- **Replier**

  rti::request::Replier *(p. 2168), SimpleReplier and associated elements*

6.33.1 Detailed Description

Support for the request-reply communication pattern.

There are two basic entities that enable this pattern:

- **rti::request::Requester** (p. 2187)
- **rti::request::Replier** (p. 2168) (and a simpler version SimpleReplier)

This functionality is built on top of RTI Connext.

A Requester publishes a request topic and subscribes to a reply topic. A Replier subscribes to the request topic and publishes the reply topic.

You can find more information about this pattern in `Request-Reply`, in the Core Libraries User's Manual.

See also

Request-Reply Examples (p. 344).

6.34 Requester

rti::request::Requester (p. 2187) and associated elements

Classes

- class **rti::request::Requester**< **RequestType, ReplyType** >

  <<***reference-type***>> *(p. 351) Allows sending requests and receiving replies*
- class **rti::request::RequesterParams**

  Contains the parameters for creating a ***rti::request::Requester*** *(p. 2187).*

6.34.1 Detailed Description

rti::request::Requester (p. 2187) and associated elements

6.35 Replier

rti::request::Replier (p. 2168), SimpleReplier and associated elements

Classes

- class **rti::request::Replier**< **RequestType, ReplyType** >
  <<***reference-type***>> *(p. 351) Allows receiving requests and sending replies*
- class **rti::request::ReplierListener**< **RequestType, ReplyType** >
  *Called when a **Replier** (p. 2168) has new available requests.*
- class **rti::request::ReplierParams**
  *Contains the parameters for creating a **rti::request::Replier** (p. 2168).*
- class **rti::request::SimpleReplier**< **RequestType, ReplyType** >
  <<***reference-type***>> *(p. 351) A callback-based replier*

6.35.1 Detailed Description

rti::request::Replier (p. 2168), SimpleReplier and associated elements

6.36 Queuing Pattern

Support for the queuing communication pattern.

Modules

- **QueueProducer**
  QueueProducer and associated elements .
- **QueueConsumer**
  QueueConsumer and associated elements .
- **QueueRequester**
  QueueRequester and associated elements .
- **QueueReplier**
  QueueReplier and associated elements .

6.36.1 Detailed Description

Support for the queuing communication pattern.

The queuing communication patterns provides a load-balancing communication model in which a sample sent by a QueueProducer is received exactly by one QueueConsumer.

This pattern requires running RTI Queuing Service.

If there are no QueueConsumers available when the sample is sent, the sample will be stored in a SharedReaderQueue hosted by RTI Queuing Service until a QueueConsumer is available to process the sample.

If a QueueConsumer receives a sample and does not acknowledge it before a specified amount of time or it acknowledges it negatively, the sample will be redelivered to a different QueueConsumer.

A SharedReaderQueue is the result of the association of a Topic with a SharedSubscriber, where a SharedSubscriber is a container that hosts SharedReaderQueues within RTI Queuing Service.

The queuing API implements four basic entities:

- QueueProducer
- QueueConsumer
- QueueRequester
- QueueReplier

Note

It is possible to include all the Queuing API at once with `#include <rti/queuing/rtiqueuing.hpp>` or by including each header file individually.

A QueueProducer sends samples to a Queuing Service instance, which stores them in a SharedReaderQueue.

A QueueConsumer receives samples from a SharedReaderQueue hosted by a Queuing Service instance.

QueueProducers and QueueConsumers do not communicate directly; they need the mediation of Queuing Service.

QueueProducers and QueueConsumers communicate with Queuing Service using different Topics.

A QueueProducer has an underlying DataWriter that publishes samples on Topic 'aTopicName'. This is the SharedReaderQueue Topic.

A QueueConsumer has an underlying DataReader that receives samples on Topic 'aTopicName@aSharedSubscriberName'.

QueueProducer and QueueConsumer are simple components that provide basic queuing communication in one direction: from QueueProducer to QueueConsumer.

The RTI Connext DDS Queuing API also supports the request-reply use case, in which samples sent by a QueueRequester are expected to generate a reply from the QueueReplier that consumed the request.

The QueueReplier produces a reply for each request that it consumes. Queuing Service correlates and delivers that reply to the QueueRequester that produced the request.

QueueRequester and QueueReplier are the components that provide the request-reply functionality. QueueRequesters and QueueRepliers do not communicate directly. They always require the mediation of Queuing Service:

- A QueueRequester sends requests to a request SharedReaderQueue and receives a reply for each request from a reply SharedReaderQueue.
- A QueueReplier receives requests from a request SharedReaderQueue and sends a reply for each request to a reply SharedReaderQueue.

QueueRequesters and QueueRepliers communicate with Queuing Service through regular SharedReaderQueues. The communication from a QueueRequester to a QueueReplier is identical to the basic queuing use-case for Producers and Consumers. The communication from a QueueReplier to a QueueRequester is also similar, with the key difference that replies from the reply SharedReaderQueue are only delivered to the QueueRequester that originally sent the associated request.

The queuing request-reply pattern can also be implemented by using the simple QueueProducer and QueueConsumer components. In this case, your application must create the necessary QueueProducers and QueueConsumers to communicate with the request and reply SharedReaderQueues, as well as perform the correlation of the request and reply samples.

All the available entities and associated infrastructure are built on top of RTI Connext. You can find detailed information about this pattern and the DDS mapping in the RTI Queuing Service User's Manual.

6.37 QueueProducer

QueueProducer and associated elements .

Classes

- class **rti::queuing::QueueProducerParams**

  *Contains the parameters for creating a **QueueProducer** (p. 2080).*
- class **rti::queuing::QueueProducer**< **T** >

  Allows you to send samples to a specific SharedReaderQueue hosted by Queuing Service.
- class **rti::queuing::QueueProducerListener**< **T** >

  *Called when certain events occur in a **QueueProducer** (p. 2080).*
- class **rti::queuing::NoOpQueueProducerListener**< **T** >

  A listener with an empty implementation of all methods.

6.37.1 Detailed Description

QueueProducer and associated elements .

6.38 QueueConsumer

QueueConsumer and associated elements .

Classes

- class **rti::queuing::QueueConsumer< T >**

  Allows you to receive samples from a SharedReaderQueue.
- class **rti::queuing::QueueConsumerListener< T >**

  *Called when certain events occur in a **QueueConsumer** (p. 2063).*
- class **rti::queuing::NoOpQueueConsumerListener< T >**

  A listener with an empty implementation of all methods.
- class **rti::queuing::QueueConsumerParams**

  *Contains the parameters for creating a **QueueConsumer** (p. 2063).*
- class **rti::queuing::ConsumerAvailabilityParams**

  Definition of the availability feedback information that can be provided by consumers to Queuing Service.

6.38.1 Detailed Description

QueueConsumer and associated elements .

6.39 QueueRequester

QueueRequester and associated elements .

Classes

- class **rti::queuing::QueueRequesterParams**

  *Contains the parameters for creating a **QueueRequester** (p. 2111).*
- class **rti::queuing::QueueRequester< TReq, TRep >**

  Allows sending requests and receiving replies.
- class **rti::queuing::QueueRequesterListener< TReq, TRep >**

  *Called when certain events occur in a **QueueRequester** (p. 2111).*
- class **rti::queuing::NoOpQueueRequesterListener< TReq, TRep >**

  A listener with an empty implementation of all methods.

6.39.1 Detailed Description

QueueRequester and associated elements .

6.40 QueueReplier

QueueReplier and associated elements .

Classes

- class **rti::queuing::QueueReplierParams**

  *Contains the parameters for creating a **QueueReplier** (p. 2094).*
- class **rti::queuing::QueueReplier< TReq, TRep >**

  Allows receiving requests and sending replies.
- class **rti::queuing::QueueReplierListener< TReq, TRep >**

  *Called when certain events occur in a **QueueReplier** (p. 2094).*
- class **rti::queuing::NoOpQueueReplierListener< TReq, TRep >**

  A listener with an empty implementation of all methods.

6.40.1 Detailed Description

QueueReplier and associated elements .

6.41 Remote Procedure Call

Remote Procedure call (RPC) communication pattern.

Modules

- **Client-side API**

  Part of the RPC API that relates to the client.
- **Server-side API**

  *Part of the RPC API that relates to the **Server** (p. 2346) and **ServiceEndpoint** (p. 2354).*

Classes

- class **dds::rpc::RemoteUnknownOperationError**

  Thrown when a Client calls an operation that doesn't exist in the Service.
- class **dds::rpc::RemoteUnsupportedOperationError**

  Thrown when a Client calls an operation that is not supported by the Service.
- class **dds::rpc::RemoteInvalidArgumentError**

  Thrown when a Client provides an invalid argument to a Service operation.
- class **dds::rpc::RemoteOutOfResourcesError**

  Thrown when the Service runs out of resources to fulfill a Client's operation.
- class **dds::rpc::RemoteUnknownExceptionError**

  Thrown when an unknown exception occurs in the Service.
- class **rpc_example::RobotControl**

  *The synchronous interface generated from the **RobotControl** (p. 97) IDL service.*
- class **rpc_example::RobotControlAsync**

  *The asynchronous interface derived from the **RobotControl** (p. 2213) service.*

6.41.1 Detailed Description

Remote Procedure call (RPC) communication pattern.

With RPC you can write client and service applications that make and process function calls according to interfaces defined in IDL. The clients and the services communicate using DDS topics.

Note

To get started follow the **RPC Tutorial** (p. 421).

The classes involved in RPC include types generated by `rtiddsgen` and supporting types in the `dds::rpc` and `rti::rpc` namespaces. This API reference assumes that an example IDL service interface is defined as follows:

```
exception WalkError {
    string<32> message;
};
exception TooFastError {
};
@final
struct Coordinates {
    int32 x;
    int32 y;
};
@service
interface RobotControl {
    Coordinates walk_to(Coordinates destination, float speed)
            raises(WalkError, TooFastError);
    float get_speed();
    attribute string<128> name;
};
```

From this IDL the following types are generated:

- **rpc_example::RobotControl** (p. 2213) (a pure-virtual abstract class)
- **rpc_example::RobotControlAsync** (p. 2215) (a pure-virtual abstract class)
- **rpc_example::RobotControlService** (p. 408) (an instantiation of **dds::rpc::ServiceEndpoint** (p. 2354))
- **rpc_example::RobotControlClient** (p. 2217) (derived from **dds::rpc::ClientEndpoint** (p. 925))

In addition, the class **dds::rpc::Server** (p. 2346) provides the execution context for one or more services.

See also

RPC Tutorial (p. 421)

6.42 Utilities

Utilities for the RTI Connext Messaging module.

Utilities for the RTI Connext Messaging module.

Connext Messaging Utilities

6.43 Durability and Persistence

APIs related to RTI Connext Durability and Persistence.

APIs related to RTI Connext Durability and Persistence.

RTI Connext offers the following mechanisms for achieving durability and persistence:

- **Durable Writer History** (p. 99)
- **Durable Reader State** (p. 99)
- **Data Durability** (p. 100)

To use the first two features, you need a relational database, which is not included with RTI Connext. Supported databases are listed in the `Release Notes`.

The third feature, provided by RTI Persistence Service, can use the filesystem or a relational database to persist information.

These three features can be used separately or in combination.

6.43.1 Durable Writer History

This feature allows a **dds::pub::DataWriter** (p. 1138) to locally persist its local history cache so that it can survive shutdowns, crashes and restarts. When an application restarts, each **dds::pub::DataWriter** (p. 1138) that has been configured to have durable writer history automatically loads all the data in its history cache from disk and can carry on sending data as if it had never stopped executing. To the rest of the system, it will appear as if the **dds::pub::DataWriter** (p. 1138) had been temporarily disconnected from the network and then reappeared.

See also

Configuring Durable Writer History (p. 101)

6.43.2 Durable Reader State

This feature allows a **dds::sub::DataReader** (p. 976) to locally persists its state and remember the sequence numbers it has already received. When an application restarts, each **dds::sub::DataReader** (p. 976) that has been configured to have durable reader state automatically loads its state from disk and can carry on receiving data as if it had never stopped executing. Data that had already been received by the **dds::sub::DataReader** (p. 976) before the restart will will not be provided to the application again.

See also

Configuring Durable Reader State (p. 102)

6.43.3 Data Durability

This feature is a full implementation of the OMG DDS Persistence Profile. The **DURABILITY** (p. 529) QoS lets an application configure a **dds::pub::DataWriter** (p. 1138) such that the information written by the **dds::pub::DataWriter** (p. 1138) survives beyond the lifetime of the **dds::pub::DataWriter** (p. 1138). In this manner, a late-joining **dds::sub::DataReader** (p. 976) can subscribe and receive the information even after the **dds::pub::DataWriter** (p. 1138) application is no longer executing. To use this feature, you need RTI Persistence Service – an optional product that can be purchased separately.

6.43.4 Durability and Persistence Based on Virtual GUID

Every modification to the global dataspace made by a **dds::pub::DataWriter** (p. 1138) is identified by a pair (virtual GUID, sequence number).

- The virtual GUID (Global Unique Identifier) is a 16-byte character identifier associated with a **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976); it is used to uniquely identify this entity in the global data space.
- The sequence number is a 64-bit identifier that identifies changes published by a specific **dds::pub::DataWriter** (p. 1138).

Several **dds::pub::DataWriter** (p. 1138) entities can be configured with the same virtual GUID. If each of these **dds::pub::DataWriter** (p. 1138) entities publishes a sample with sequence number '0', the sample will only be received once by the **dds::sub::DataReader** (p. 976) entities subscribing to the content published by the **dds::pub::DataWriter** (p. 1138) entities.

RTI Connext also uses the virtual GUID (Global Unique Identifier) to associate a persisted state (state in permanent storage) to the corresponding DDS entity.

For example, the history of a **dds::pub::DataWriter** (p. 1138) will be persisted in a database table with a name generated from the virtual GUID of the **dds::pub::DataWriter** (p. 1138). If the **dds::pub::DataWriter** (p. 1138) is restarted, it must have associated the same virtual GUID to restore its previous history.

Likewise, the state of a **dds::sub::DataReader** (p. 976) will be persisted in a database table whose name is generated from the **dds::sub::DataReader** (p. 976) virtual GUID

A **dds::pub::DataWriter** (p. 1138)'s virtual GUID can be configured using **rti::core::policy::DataWriterProtocol::virtual_guid** (p. 1212). Similarly, a **dds::sub::DataReader** (p. 976)'s virtual GUID can be configured using **rti::core::policy::DataReaderProtocol::virtual_guid** (p. 1060).

The **dds::topic::PublicationBuiltinTopicData** (p. 1977) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428) structures include the virtual GUID associated with the discovered publication or subscription.

Refer to the `User's Manual` for additional use cases.

See also

rti::core::policy::DataWriterProtocol::virtual_guid (p. 1212) **rti::core::policy::DataReaderProtocol::virtual_guid** (p. 1060).

6.43.5 Configuring Durable Writer History

To configure a **dds::pub::DataWriter** (p. 1138) to have durable writer history, use the **PROPERTY** (p. 541) QoS policy associated with the **dds::pub::DataWriter** (p. 1138) or the **dds::domain::DomainParticipant** (p. 1326).

Properties defined for the **dds::domain::DomainParticipant** (p. 1326) will be applied to all the **dds::pub::DataWriter** (p. 1138) objects belonging to the **dds::domain::DomainParticipant** (p. 1326), unless the property is overwritten by the **dds::pub::DataWriter** (p. 1138).

See also

rti::core::policy::Property (p. 1969)

The following table lists the supported durable writer history properties.

Table 6.9 ***Durable Writer History Properties***

Property	Description
dds.data_writer.history.plugin_name	Must be set to "dds.data_writer.history.odbc_plugin.↩ builtin" to enable durable writer history in the DataWriter. This property is required.
dds.data_writer.history.odbc_plugin.builtin.dsn	The ODBC DSN (Data Source Name) associated with the database where the writer history must be persisted. This property is required.
dds.data_writer.history.odbc_plugin.builtin.driver	This property tells RTI Connext which ODBC driver to load. If the property is not specified, RTI Connext will try to use the standard ODBC driver manager library↩ : UnixOdbc (odbc32.dll) on Linux systems; the Windows ODBC driver manager (libodbc.so) on Windows systems.
dds.data_writer.history.odbc_plugin.builtin.username	Configures the username used to connect to the database. This property is not used if it is unspecified. There is no default value.
dds.data_writer.history.odbc_plugin.builtin.password	Configures the password used to connect to the database. This property is not used if it is unspecified. There is no default value.
dds.data_writer.history.odbc_plugin.builtin.shared	If set to 1, RTI Connext creates a single connection per DSN that will be shared across DataWriters within the same Publisher. If set to 0 (the default), a **dds::pub↩ ::DataWriter** (p. 1138) will create its own database connection. Default: 0
dds.data_writer.history.odbc_plugin.builtin.instance_↩ cache_max_size	These properties configure the resource limits associated with the ODBC writer history caches. To minimize the number of accesses to the database, RTI Connext uses two caches, one for samples and one for instances. The initial and maximum sizes of these caches are configured using these properties. The resource limits initial_instances, max_instances, initial_samples, max_samples and max_samples_per_instance in the **dds::core::policy::ResourceLimits** (p. 2203) are used to configure the maximum number of samples and instances that can be stored in the relational database. Default: **dds::core::policy::ResourceLimits::max_↩ instances** (p. 2206)

Property	Description
dds.data_writer.history.odbc_plugin.builtin.instance_↩ cache_init_size	See description above. Default: **dds::core::policy::↩ ResourceLimits::initial_instances** (p. 2208)
dds.data_writer.history.odbc_plugin.builtin.sample_↩ cache_max_size	See description above. Default: 32 (the minimum)
dds.data_writer.history.odbc_plugin.builtin.sample_↩ cache_init_size	See description above. Default: 32
dds.data_writer.history.odbc_plugin.builtin.restore	This property indicates whether or not the persisted writer history must be restored once the **dds::pub::↩ DataWriter** (p. 1138) is restarted. If the value is 0, the content of the database associated with the **dds::pub↩ ::DataWriter** (p. 1138) being restarted will be deleted. If the value is 1, the **dds::pub::DataWriter** (p. 1138) will restore its previous state from the database content. Default: 1
dds.data_writer.history.odbc_plugin.builtin.in_memory↩ _state	This property determines how much state will be kept in memory by the ODBC writer history in order to avoid accessing the database. When in_memory_state is equal to 1, instance_cache↩ _max_size is always equal to **dds::core::policy::↩ ResourceLimits::max_instances** (p. 2206) (it cannot be changed). In addition, the ODBC writer history will keep in memory a fixed state overhead of 24 bytes per sample. In this operating mode, the ODBC writer history provides the best performance. However, the restore operation will be slower and the maximum number of samples that the writer history can manage will be limited by the available physical memory. If in_memory_state is equal to 0, all the state will be kept in the underlying database. In this operating mode, the maximum number of samples in the writer history will not be limited by the physical memory available unless the underlying database is an in-memory database (Times↩ Ten). Default: 1

6.43.6 Configuring Durable Reader State

To configure a **dds::sub::DataReader** (p. 976) with durable reader state, use the **PROPERTY** (p. 541) QoS policy associated with the **dds::sub::DataReader** (p. 976) or **dds::domain::DomainParticipant** (p. 1326).

A property defined in the **dds::domain::DomainParticipant** (p. 1326) will be applicable to all the **dds::sub::Data↩ Reader** (p. 976) belonging to the **dds::domain::DomainParticipant** (p. 1326) unless it is overwritten by the **dds::sub↩ ::DataReader** (p. 976).

See also

rti::core::policy::Property (p. 1969)

The following table lists the supported durable reader state properties.

Table 6.10 ***Durable Reader State Properties***

Property	Description
dds.data_reader.state.odbc.dsn	The ODBC DSN (Data Source Name) associated with the database where the **dds::sub::DataReader** (p. 976) state must be persisted. This property is required.
dds.data_reader.state.filter_redundant_samples	To enable durable reader state, this property must be set to 1. Otherwise, the reader state will not be kept and/or persisted. When the reader state is not maintained, RTI Connext does not filter duplicate samples that may be coming from the same virtual writer. By default, this property is set to 1.
dds.data_reader.state.odbc.driver	This property is used to indicate which ODBC driver to load. If the property is not specified, RTI Connext will try to use the standard ODBC driver manager library↩ : UnixOdbc (odbc32.dll) on Linux systems; the Windows ODBC driver manager (libodbc.so) on Windows systems).
dds.data_reader.state.odbc.username	This property configures the username used to connect to the database. This property is not used if it is unspecified. There is no default value.
dds.data_reader.state.odbc.password	This property configures the password used to connect to the database. This property is not used if it is unspecified. There is no default value.
dds.data_reader.state.restore	This property indicates if the persisted **dds::sub::Data↩ Reader** (p. 976) state must be restored or not once the **dds::sub::DataReader** (p. 976) is restarted. If this property is 0, the previous state will be deleted from the database. If it is 1, the **dds::sub::DataReader** (p. 976) will restore its previous state from the database content. Default: 1
dds.data_reader.state.checkpoint_frequency	This property controls how often the reader state is stored in the database. A value of N means to store the state once every N samples. A high value will provide better performance. However, if the reader is restarted it may receive some duplicate samples. These samples will be filtered by the middleware and they will not be propagated to the application. Default: 1
dds.data_reader.state.persistence_service.request_↩ depth	This property indicates the number of most recent historical samples that the persisted **dds::sub::DataReader** (p. 976) wants to receive when it starts up. Default: 0

6.43.7 Configuring Data Durability

RTI Connext implements **dds::core::policy::DurabilityKind_def::TRANSIENT** (p. 1442) and **dds::core::policy::↩ DurabilityKind_def::PERSISTENT** (p. 1442) durability using RTI Persistence Service, available for purchase as a separate RTI product.

For more information on RTI Persistence Service, refer to the `User's Manual`, or the RTI Persistence Service API Reference HTML documentation.

See also

DURABILITY (p. 529)

6.44 Basic QoS

QoS that are considered basic and fundamental to RTI Connext.

QoS that are considered basic and fundamental to RTI Connext.

This page lists all QoS that are considered Basic. Basic QoS are applicable to many different use cases, are widely applicable to almost any customer, and should be relatively easy to understand and set.

These are QoS policies and fields that you should be aware of even if you are not setting them today. These QoS are fundamental to DDS and to RTI Connext, and showcase RTI Connext capabilities and features.

All other Qos are applicable to a smaller subset of use cases and customers, or are finer-grained controls for a Basic QoS.

dds::core::policy::Durability (p. 1433)	**kind** (p. 1437)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of durability.
dds::core::policy::Durability (p. 1433)	**writer_depth** (p. 1439)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Indicates the number of samples per instance that a durable **dds::pub::DataWriter** (p. 1138) will send to a late-joining **dds::sub::DataReader** (p. 976).
dds::core::policy::DataRepresentation (p. 1113)	**compression_settings** (p. 1117)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Structure that contains the compression settings.
dds::core::policy::Deadline (p. 1255)	**period** (p. 1258)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Duration of the deadline period.
dds::core::policy::Ownership (p. 1899)	**kind** (p. 1904)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of ownership
dds::core::policy::OwnershipStrength (p. 1906)	**value** (p. 1908)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The strength value used to arbitrate among multiple writers.
dds::core::policy::Liveliness (p. 1645)	**kind** (p. 1649)

	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of liveliness desired.
dds::core::policy::Liveliness (p. 1645)	**lease_duration** (p. 1649)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The duration within which a **dds::pub::DataWriter** (p. 1138) must be asserted, or else it is assumed to be not alive.
dds::core::policy::Liveliness (p. 1645)	**assertions_per_lease_duration** (p. 1651)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The number of assertions a **dds::pub::DataWriter** (p. 1138) will send during a its **dds::core::policy::Liveliness↩ ::lease_duration** (p. 1649).
dds::core::policy::TimeBasedFilter (p. 2469)	**minimum_separation** (p. 2472)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The minimum separation duration between subsequent samples.
dds::core::policy::Partition (p. 1923)	**name** (p. 1926)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* A list of partition names
dds::core::policy::Reliability (p. 2149)	**kind** (p. 2153)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Kind of reliability.
dds::core::policy::Reliability (p. 2149)	**max_blocking_time** (p. 2153)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The maximum time a DataWriter can block on a **write()** (p. 1179) call.
dds::core::policy::DestinationOrder (p. 1258)	**kind** (p. 1261)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Specifies the desired kind of destination order.
dds::core::policy::History (p. 1602)	**kind** (p. 1604)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Specifies the kind of history to be kept.
dds::core::policy::History (p. 1602)	**depth** (p. 1605)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Specifies the number of samples per instance to be kept, when the `kind` is **dds::core::policy::HistoryKind_def::KEEP↩ _LAST** (p. 1607)
dds::core::policy::ResourceLimits (p. 2203)	**max_samples** (p. 2205)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Represents the maximum samples the middleware can store for any one **dds::pub::DataWriter** (p. 1138) (or **dds::sub::↩ DataReader** (p. 976)).

dds::core::policy::ResourceLimits (p. 2203)	**max_instances** (p. 2206)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Represents the maximum number of instances a **dds::pub::Data↩ Writer** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::ResourceLimits (p. 2203)	**initial_samples** (p. 2207)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Represents the initial samples the middleware will store for any one **dds::pub::DataWriter** (p. 1138) (or **dds::sub↩ ::DataReader** (p. 976)).
dds::core::policy::ResourceLimits (p. 2203)	**initial_instances** (p. 2208)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Represents the initial number of instances a **dds↩ ::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) will manage.
dds::core::policy::EntityFactory (p. 1527)	**autoenable_created_entities** (p. 1529)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Specifies whether the entity acting as a factory automatically enables the instances it creates.
dds::core::policy::Lifespan (p. 1635)	**duration** (p. 1637)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Maximum duration for the data's validity.
rti::core::policy::PublishMode (p. 2013)	**kind** (p. 2016)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Publishing mode
rti::core::policy::WireProtocol (p. 2630)	**participant_id** (p. 2635)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* A value used to distinguish among different participants belonging to the same domain on the same host.
rti::core::policy::TransportBuiltin (p. 2533)	**mask** (p. 2536)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Specifies the built-in transports that are registered automatically when the **dds::domain::DomainParticipant** (p. 1326) is enabled.
rti::core::policy::TransportMulticast (p. 2543)	**value** (p. 2545)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* A sequence of multicast communications settings.
rti::core::policy::Discovery (p. 1265)	**initial_peers** (p. 1268)

	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.
rti::core::policy::Discovery (p. 1265)	**multicast_receive_addresses** (p. 1268)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the Domain↩ Participant.
rti::core::policy::Discovery (p. 1265)	**accept_unknown_peers** (p. 1270)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Whether to accept a new participant that is not in the initial peers list.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_liveliness_lease_duration** (p. 1279)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The liveliness lease duration for the participant.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_liveliness_assert_period** (p. 1279)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The period to assert liveliness for the participant.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_announcement_period** (p. 1280)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
rti::core::policy::DiscoveryConfig (p. 1271)	**initial_participant_announcements** (p. 1282)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The number of initial announcements sent when a participant is first enabled.
rti::core::policy::DiscoveryConfig (p. 1271)	**new_remote_participant_announcements** (p. 1283)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The number of participant announcements sent when a remote participant is newly discovered.
rti::core::policy::DiscoveryConfig (p. 1271)	**min_initial_participant_announcement_period** (p. 1283)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
rti::core::policy::DiscoveryConfig (p. 1271)	**max_initial_participant_announcement_period** (p. 1284)

	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**heartbeat_period** (p. 2234)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The period at which to send heartbeats.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**fast_heartbeat_period** (p. 2235)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**late_joiner_heartbeat_period** (p. 2236)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**min_send_window_size** (p. 2249)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Minimum size of send window of unacknowledged samples
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_send_window_size** (p. 2250)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Maximum size of send window of unacknowledged samples
rti::core::policy::Property (p. 1969)	**value** (p. 1973)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sequence of properties.
rti::core::policy::EntityName (p. 1530)	**name** (p. 1531)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The name of the entity.
rti::core::policy::Batch (p. 880)	**enable** (p. 882)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Specifies whether or not batching is enabled.
rti::core::policy::Batch (p. 880)	**max_data_bytes** (p. 882)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The maximum cumulative length of all serialized samples in a batch.

rti::core::policy::Batch (p. 880)	**max_samples** (p. 883)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The maximum number of samples in a batch.
rti::core::QosProviderParams (p. 2049)	**string_profile** (p. 2050)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sequence of strings containing a XML document to load.
rti::core::QosProviderParams (p. 2049)	**url_profile** (p. 2050)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sequence of **URL groups** (p. 304) containing a set of XML documents to load.
rti::core::QosProviderParams (p. 2049)	**ignore_user_profile** (p. 2051)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Ignores the file USER_QOS_PROFILES.xml in the current working directory.
rti::core::QosProviderParams (p. 2049)	**ignore_environment_profile** (p. 2052)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Ignores the value of the **NDDS_QOS_PROFILES** (p. 304).
rti::core::QosProviderParams (p. 2049)	**ignore_resource_profile** (p. 2052)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Ignores the file NDDS_QOS_PROFILES.xml.
rti::config::Logger (p. 1683)	**verbosity** (p. 1685)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The verbosities at which RTI Connext diagnostic information is logged.
rti::config::Logger (p. 1683)	**category** (p. 1685)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Categories of logged messages.
rti::config::Logger (p. 1683)	**output_file** (p. 1686)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Specifies the file to which log messages will be redirected to.
rti::config::Logger (p. 1683)	**output_file_suffix** (p. 1687)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sets the file suffix when logging to a set of files
rti::core::MonitoringEventDistributionSettings (p. 1722)	**publication_period** (p. 1726)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.

rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**polling_period** (p. 1744)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Period at which the periodic metric thread polls and publishes the periodic metrics.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**max_historical_logs** (p. 1729)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**publication_period** (p. 1732)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Period at which the logging thread publishes log messages.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**enable** (p. 1713)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Enables the use of a dedicated **dds::domain::Domain↩ Participant** (p. 1326) to distribute the RTI Connext application telemetry data.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**domain_id** (p. 1713)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The domain ID used in the creation of RTI Monitoring Library 2.0 **dds::domain::DomainParticipant** (p. 1326).
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**collector_initial_peers** (p. 1715)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **rti::core::MonitoringDistribution↩ Settings::dedicated_participant** (p. 1719).
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**middleware_forwarding_level** (p. 1734)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Log messages with **rti::config::LogFacility::middleware** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**security_event_forwarding_level** (p. 1734)

	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Log messages with **rti::config::LogFacility::security_event** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**service_forwarding_level** (p. 1735)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Log messages with **rti::config::LogFacility::service** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**user_forwarding_level** (p. 1736)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Log messages with **rti::config::LogFacility::user** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringTelemetryData (p. 1744)	**metrics** (p. 1745)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sequence of **rti::core::MonitoringMetricSelection** (p. 1736) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.
rti::core::MonitoringTelemetryData (p. 1744)	**logs** (p. 1746)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* **rti::core::MonitoringLoggingForwardingSettings** (p. 1733) containing the **rti::config::SyslogVerbosity** (p. 458) levels that will be forwarded for the different **rti::config::LogFacility** (p. 459).
rti::core::policy::Monitoring (p. 1706)	**enable** (p. 1708)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.
rti::core::policy::Monitoring (p. 1706)	**application_name** (p. 1709)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The name of the resource that represents this RTI Connext application.
rti::core::policy::TopicQueryDispatch (p. 2522)	**publication_period** (p. 2525)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sets the periodic interval at which samples are published.

6.45 DDS Specification QoS

QoS that are defined in one of the OMG DDS Specifications.

QoS that are defined in one of the OMG DDS Specifications.

This page lists the QoS that are included in an OMG DDS specification. The standards are listed here: https://www.omg.org/spec/category/omg-data-distribution-service/.

dds::core::policy::UserData (p. 2591)	**value** (p. 2593)
	<<***spec***>> *(p. 112)* a sequence of octets
dds::core::policy::GroupData (p. 1591)	**value** (p. 1593)
	<<***spec***>> *(p. 112)* a sequence of octets
dds::core::policy::Durability (p. 1433)	**kind** (p. 1437)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* The kind of durability.
dds::core::policy::DurabilityService (p. 1443)	**service_cleanup_delay** (p. 1445)
	<<***spec***>> *(p. 112)* Controls when the service is able to remove all information regarding a data instances.
dds::core::policy::DurabilityService (p. 1443)	**history_kind** (p. 1445)
	<<***spec***>> *(p. 112)* The kind of history to apply in recouping durable data.
dds::core::policy::DurabilityService (p. 1443)	**history_depth** (p. 1446)
	<<***spec***>> *(p. 112)* Setting to use for the **dds::core::policy::Durability::writer_depth** (p. 1439) when recouping durable data.
dds::core::policy::DurabilityService (p. 1443)	**max_samples** (p. 1446)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_instances** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_samples_per_instance** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DataRepresentation (p. 1113)	**value** (p. 1116)
	<<***spec***>> *(p. 112)* Sequence of representation identifiers.
dds::core::policy::Presentation (p. 1942)	**access_scope** (p. 1947)

	<<***spec***>> *(p. 112)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
dds::core::policy::Presentation (p. 1942)	**coherent_access** (p. 1947)
	<<***spec***>> *(p. 112)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.
dds::core::policy::Presentation (p. 1942)	**ordered_access** (p. 1948)
	<<***spec***>> *(p. 112)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
dds::core::policy::Deadline (p. 1255)	**period** (p. 1258)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Duration of the deadline period.
dds::core::policy::LatencyBudget (p. 1631)	**duration** (p. 1633)
	<<***spec***>> *(p. 112)* Duration of the maximum acceptable delay.
dds::core::policy::Ownership (p. 1899)	**kind** (p. 1904)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* The kind of ownership
dds::core::policy::OwnershipStrength (p. 1906)	**value** (p. 1908)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* The strength value used to arbitrate among multiple writers.
dds::core::policy::Liveliness (p. 1645)	**kind** (p. 1649)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* The kind of liveliness desired.
dds::core::policy::Liveliness (p. 1645)	**lease_duration** (p. 1649)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* The duration within which a **dds::pub::DataWriter** (p. 1138) must be asserted, or else it is assumed to be not alive.
dds::core::policy::TimeBasedFilter (p. 2469)	**minimum_separation** (p. 2472)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* The minimum separation duration between subsequent samples.
dds::core::policy::Partition (p. 1923)	**name** (p. 1926)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* A list of partition names
dds::core::policy::Reliability (p. 2149)	**kind** (p. 2153)

	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Kind of reliability.
dds::core::policy::Reliability (p. 2149)	**max_blocking_time** (p. 2153)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The maximum time a DataWriter can block on a **write()** (p. 1179) call.
dds::core::policy::DestinationOrder (p. 1258)	**kind** (p. 1261)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Specifies the desired kind of destination order.
dds::core::policy::DestinationOrder (p. 1258)	**scope** (p. 1262)
	*<<**spec**>> (p. 112)* Specifies the desired scope of the source destination order.
dds::core::policy::History (p. 1602)	**kind** (p. 1604)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Specifies the kind of history to be kept.
dds::core::policy::History (p. 1602)	**depth** (p. 1605)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Specifies the number of samples per instance to be kept, when the `kind` is **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607)
dds::core::policy::ResourceLimits (p. 2203)	**max_samples** (p. 2205)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Represents the maximum samples the middleware can store for any one **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)).
dds::core::policy::ResourceLimits (p. 2203)	**max_instances** (p. 2206)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Represents the maximum number of instances a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::ResourceLimits (p. 2203)	**max_samples_per_instance** (p. 2207)
	*<<**spec**>> (p. 112)* Represents the maximum number of samples of any one instance a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::EntityFactory (p. 1527)	**autoenable_created_entities** (p. 1529)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Specifies whether the entity acting as a factory automatically enables the instances it creates.
dds::core::policy::TransportPriority (p. 2552)	**value** (p. 2553)

	<<***spec***>> *(p. 112)* This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.
dds::core::policy::Lifespan (p. 1635)	**duration** (p. 1637)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Maximum duration for the data's validity.
dds::core::policy::WriterDataLifecycle (p. 2660)	**autodispose_unregistered_instances** (p. 2661)
	<<***spec***>> *(p. 112)* Boolean flag that controls whether or not a writer will also dispose of an instance when the **dds::pub::DataWriter** (p. 1138) unregisters from the instance by means of the unregister operations.
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_nowriter_samples_delay** (p. 2140)
	<<***spec***>> *(p. 112)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain samples for an instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_↩ no_writers()** (p. 1617).
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_disposed_samples_delay** (p. 2141)
	<<***spec***>> *(p. 112)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain samples for an instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_↩ disposed()** (p. 1617).
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**kind** (p. 2565)
	<<***spec***>> *(p. 112)* Type consistency kind.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_sequence_bounds** (p. 2566)
	<<***spec***>> *(p. 112)* Controls whether sequence bounds are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_string_bounds** (p. 2567)
	<<***spec***>> *(p. 112)* Controls whether string bounds are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_member_names** (p. 2567)
	<<***spec***>> *(p. 112)* Controls whether member names are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**prevent_type_widening** (p. 2568)

	<<***spec***>> *(p. 112)* Controls whether type widening is allowed.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**force_type_validation** (p. 2568)
	<<***spec***>> *(p. 112)* Controls whether type information must be available in order to complete matching between a **dds::pub::DataWriter** (p. 1138) and a **dds::sub::DataReader** (p. 976).
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_enum_literal_names** (p. 2569)
	<<***spec***>> *(p. 112)* Controls whether enumeration constant names are taken into consideration for type assignability.
dds::core::policy::DataTag (p. 1132)	**tags** (p. 1137)
	<<***spec***>> *(p. 112)* Sequence of data tags.

6.46 DDS Extension QoS

QoS that are RTI extensions to the DDS standard.

QoS that are RTI extensions to the DDS standard.

This page lists all of the QoS that are RTI extensions of the DDS Standard. The extension QoS complement the standard QoS specified by the OMG DDS specifications. They are provided to improve product usability and enable access to product-specific features.

rti::core::policy::Availability (p. 868)	**enable_required_subscriptions** (p. 871)
	<<***extension***>> *(p. 116)* Enables support for required subscriptions in a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::Availability (p. 868)	**max_data_availability_waiting_time** (p. 871)
	<<***extension***>> *(p. 116)* Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
rti::core::policy::Availability (p. 868)	**max_endpoint_availability_waiting_time** (p. 872)
	<<***extension***>> *(p. 116)* Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
rti::core::policy::Availability (p. 868)	**required_matched_endpoint_groups** (p. 873)
	<<***extension***>> *(p. 116)* A sequence of endpoint groups.

dds::core::policy::Durability (p. 1433)	**direct_communication** (p. 1438)
	<<***extension***>> *(p. 116)* Indicates whether or not a TRANSIENT or PERSISTENT **dds::sub::DataReader** (p. 976) should receive samples directly from a TRANSIENT or PERSISTENT **dds::pub::DataWriter** (p. 1138)
dds::core::policy::Durability (p. 1433)	**writer_depth** (p. 1439)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Indicates the number of samples per instance that a durable **dds::pub::DataWriter** (p. 1138) will send to a late-joining **dds::sub::DataReader** (p. 976).
dds::core::policy::Durability (p. 1433)	**storage_settings** (p. 1440)
	<<***extension***>> *(p. 116)* Configures durable writer history and durable reader state.
rti::core::PersistentStorageSettings (p. 1927)	**enable** (p. 1929)
	<<***extension***>> *(p. 116)* Enables durable writer history in a **dds::pub::DataWriter** (p. 1138) and durable reader state in a **dds::sub::DataReader** (p. 976).
rti::core::PersistentStorageSettings (p. 1927)	**file_name** (p. 1929)
	<<***extension***>> *(p. 116)* The file name where the durable writer history or durable reader state will be stored.
rti::core::PersistentStorageSettings (p. 1927)	**trace_file_name** (p. 1931)
	<<***extension***>> *(p. 116)* The file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.
rti::core::PersistentStorageSettings (p. 1927)	**journal_kind** (p. 1932)
	<<***extension***>> *(p. 116)* Sets the journal mode of the persistent storage.
rti::core::PersistentStorageSettings (p. 1927)	**synchronization_kind** (p. 1933)
	<<***extension***>> *(p. 116)* Sets the level of synchronization with the physical disk.
rti::core::PersistentStorageSettings (p. 1927)	**vacuum** (p. 1933)
	<<***extension***>> *(p. 116)* Sets the auto-vacuum status of the storage.
rti::core::PersistentStorageSettings (p. 1927)	**restore** (p. 1934)
	<<***extension***>> *(p. 116)* Indicates if the persisted writer history or reader state must be restored.
rti::core::PersistentStorageSettings (p. 1927)	**writer_instance_cache_allocation** (p. 1934)

	<<***extension***>> *(*p. *116)* Configures the resource limits associated with the instance durable writer history cache.
rti::core::PersistentStorageSettings (p. 1927)	**writer_sample_cache_allocation** (p. 1935)
	<<***extension***>> *(*p. *116)* Configures the resource limits associated with the sample durable writer history cache.
rti::core::PersistentStorageSettings (p. 1927)	**writer_memory_state** (p. 1936)
	<<***extension***>> *(*p. *116)* Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.
rti::core::PersistentStorageSettings (p. 1927)	**reader_checkpoint_frequency** (p. 1937)
	<<***extension***>> *(*p. *116)* Controls how often the reader state is stored into the database.
rti::core::CompressionSettings (p. 943)	**compression_ids** (p. 945)
	<<***extension***>> *(*p. *116)* Mask that represents the compression algorithms enabled.
rti::core::CompressionSettings (p. 943)	**writer_compression_level** (p. 946)
	<<***extension***>> *(*p. *116)* The level of compression to use when compressing data.
rti::core::CompressionSettings (p. 943)	**writer_compression_threshold** (p. 947)
	<<***extension***>> *(*p. *116)* The threshold, in bytes, above which a serialized sample will be eligible to be compressed.
dds::core::policy::DataRepresentation (p. 1113)	**compression_settings** (p. 1117)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Structure that contains the compression settings.
dds::core::policy::Presentation (p. 1942)	**drop_incomplete_coherent_set** (p. 1949)
	<<***extension***>> *(*p. *116)* Indicates whether or not a **dds::sub::DataReader** (p. 976) should drop samples from an incomplete coherent set (one for which not all the samples were received). Such samples are reported as lost in the SAMPLE_LOST Status.
dds::core::policy::Liveliness (p. 1645)	**assertions_per_lease_duration** (p. 1651)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* The number of assertions a **dds::pub::DataWriter** (p. 1138) will send during a its **dds::core::policy::Liveliness**↩ **::lease_duration** (p. 1649).
dds::core::policy::Reliability (p. 2149)	**acknowledgment_kind** (p. 2154)
	<<***extension***>> *(*p. *116)* Kind of reliable acknowledgment

dds::core::policy::Reliability (p. 2149)	**instance_state_consistency_kind** (p. 2155)
	<<***extension***>> *(p. 116)* Whether instance state consistency is enabled
dds::core::policy::Reliability (p. 2149)	**exact_reliability_kind_match** (p. 2155)
	<<***extension***>> *(p. 116)* Whether an exact reliability kind match is required for endpoint matching.
dds::core::policy::DestinationOrder (p. 1258)	**source_timestamp_tolerance** (p. 1262)
	<<***extension***>> *(p. 116)* Allowed tolerance between source timestamps of consecutive samples.
dds::core::policy::ResourceLimits (p. 2203)	**initial_samples** (p. 2207)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Represents the initial samples the middleware will store for any one **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)).
dds::core::policy::ResourceLimits (p. 2203)	**initial_instances** (p. 2208)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Represents the initial number of instances a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) will manage.
dds::core::policy::ResourceLimits (p. 2203)	**instance_hash_buckets** (p. 2208)
	[DEPRECATED] (p. **296)** <<***extension***>> *(p. 116)* Number of hash buckets for instances.
rti::core::policy::TypeSupport (p. 2574)	**plugin_data** (p. 2575)
	[DEPRECATED] (p. **296)** <<***extension***>> *(p. 116)* Value to pass into the type plugin's de-/serialization function
rti::core::policy::TypeSupport (p. 2574)	**cdr_padding_kind** (p. 2576)
	[DEPRECATED] (p. **296)** <<***extension***>> *(p. 116)* Determines whether or not the padding bytes will be set to zero during CDR serialization.
rti::core::policy::PublishMode (p. 2013)	**kind** (p. 2016)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Publishing mode
rti::core::policy::PublishMode (p. 2013)	**flow_controller_name** (p. 2016)
	<<***extension***>> *(p. 116)* Name of the associated flow controller
rti::core::policy::PublishMode (p. 2013)	**priority** (p. 2017)
	<<***extension***>> *(p. 116)* Publication priority

rti::core::policy::AsynchronousPublisher (p. 832)	**disable_asynchronous_write** (p. 835)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Disable asynchronous publishing.
rti::core::policy::AsynchronousPublisher (p. 832)	**thread** (p. 835)
	<<***extension***>> *(p. 116)* Settings of the publishing thread.
rti::core::policy::AsynchronousPublisher (p. 832)	**disable_asynchronous_batch** (p. 836)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Disable asynchronous batch flushing.
rti::core::policy::AsynchronousPublisher (p. 832)	**asynchronous_batch_thread** (p. 837)
	<<***extension***>> *(p. 116)* Settings of the batch flushing thread.
rti::core::policy::AsynchronousPublisher (p. 832)	**disable_topic_query_publication** (p. 838)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Disable topic query publication.
rti::core::policy::AsynchronousPublisher (p. 832)	**topic_query_publication_thread** (p. 838)
	<<***extension***>> *(p. 116)* Settings of the **rti::sub::TopicQuery** (p. 2516) publication thread.
dds::core::policy::WriterDataLifecycle (p. 2660)	**autopurge_unregistered_instances_delay** (p. 2662)
	<<***extension***>> *(p. 116)* Maximum duration for which the **dds::pub::DataWriter** (p. 1138) will maintain information regarding an instance once it has unregistered the instance.
dds::core::policy::WriterDataLifecycle (p. 2660)	**autopurge_disposed_instances_delay** (p. 2664)
	<<***extension***>> *(p. 116)* Maximum duration for which the **dds::pub::DataWriter** (p. 1138) will maintain information regarding an instance once it has disposed the instance.
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_disposed_instances_delay** (p. 2142)
	<<***extension***>> *(p. 116)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain information about a received instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) and there are no samples for the instance in the DataReader queue.
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_nowriter_instances_delay** (p. 2143)
	<<***extension***>> *(p. 116)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain information about a received instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) and there are no samples for the instance in the DataReader queue.

rti::core::policy::WireProtocol (p. 2630)	**participant_id** (p. 2635)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* A value used to distinguish among different participants belonging to the same domain on the same host.
rti::core::policy::WireProtocol (p. 2630)	**rtps_host_id** (p. 2636)
	<<***extension***>> *(p. 116)* The RTPS Host ID of the domain participant.
rti::core::policy::WireProtocol (p. 2630)	**rtps_app_id** (p. 2636)
	<<***extension***>> *(p. 116)* The RTPS App ID of the domain participant.
rti::core::policy::WireProtocol (p. 2630)	**rtps_instance_id** (p. 2637)
	<<***extension***>> *(p. 116)* The RTPS Instance ID of the **dds::domain::DomainParticipant** (p. 1326).
rti::core::policy::WireProtocol (p. 2630)	**rtps_well_known_ports** (p. 2638)
	<<***extension***>> *(p. 116)* Configures the RTPS well-known port mappings.
rti::core::policy::WireProtocol (p. 2630)	**rtps_reserved_port_mask** (p. 2639)
	<<***extension***>> *(p. 116)* Specifies which well-known ports to reserve when enabling the participant.
rti::core::policy::WireProtocol (p. 2630)	**rtps_auto_id_kind** (p. 2639)
	<<***extension***>> *(p. 116)* Kind of auto mechanism used to calculate the GUID prefix.
rti::core::policy::WireProtocol (p. 2630)	**compute_crc** (p. 2640)
	<<***extension***>> *(p. 116)* Adds RTPS CRC submessage to every message when this field is set to true.
rti::core::policy::WireProtocol (p. 2630)	**check_crc** (p. 2640)
	<<***extension***>> *(p. 116)* Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to true.
rti::core::policy::TransportSelection (p. 2554)	**enabled_transports** (p. 2555)
	<<***extension***>> *(p. 116)* A sequence of transport aliases that specifies the transport instances available for use by the entity.
rti::core::policy::TransportBuiltin (p. 2533)	**mask** (p. 2536)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Specifies the built-in transports that are registered automatically when the **dds::domain::DomainParticipant** (p. 1326) is enabled.

rti::core::policy::TransportUnicast (p. 2556)	**value** (p. 2558)
	<<***extension***>> *(p. 116)* A sequence of unicast communication settings.
rti::core::policy::TransportMulticastMapping (p. 2547)	**value** (p. 2549)
	<<***extension***>> *(p. 116)* A sequence of multicast communication mappings.
rti::core::policy::TransportMulticast (p. 2543)	**value** (p. 2545)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* A sequence of multicast communications settings.
rti::core::policy::TransportMulticast (p. 2543)	**kind** (p. 2546)
	<<***extension***>> *(p. 116)* A value that specifies a way to determine how to obtain the multicast address.
rti::core::policy::Discovery (p. 1265)	**enabled_transports** (p. 1267)
	<<***extension***>> *(p. 116)* The transports available for use by the Discovery mechanism.
rti::core::policy::Discovery (p. 1265)	**initial_peers** (p. 1268)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.
rti::core::policy::Discovery (p. 1265)	**multicast_receive_addresses** (p. 1268)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the Domain↩ Participant.
rti::core::policy::Discovery (p. 1265)	**metatraffic_transport_priority** (p. 1269)
	<<***extension***>> *(p. 116)* The transport priority to use for the Discovery meta-traffic.
rti::core::policy::Discovery (p. 1265)	**accept_unknown_peers** (p. 1270)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Whether to accept a new participant that is not in the initial peers list.
rti::core::policy::Discovery (p. 1265)	**enable_endpoint_discovery** (p. 1271)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Whether to automatically enable endpoint discovery for all the remote participants.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_writers** (p. 1085)

	<<***extension***>> *(p. 116)* The maximum number of remote writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_writers_per_instance** (p. 1086)
	<<***extension***>> *(p. 116)* The maximum number of remote writers from which a **dds::sub::DataReader** (p. 976) may read a single instance.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_samples_per_remote_writer** (p. 1087)
	<<***extension***>> *(p. 116)* The maximum number of out-of-order samples from a given remote **dds::pub::DataWriter** (p. 1138) that a **dds::sub::DataReader** (p. 976) may store when maintaining a reliable connection to the **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_infos** (p. 1087)
	<<***extension***>> *(p. 116)* The maximum number of info units that a **dds::sub::DataReader** (p. 976) can use to store **dds::sub::SampleInfo** (p. 2285).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_remote_writers** (p. 1088)
	<<***extension***>> *(p. 116)* The initial number of remote writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_remote_writers_per_instance** (p. 1089)
	<<***extension***>> *(p. 116)* The initial number of remote writers from which a **dds::sub::DataReader** (p. 976) may read a single instance.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_infos** (p. 1089)
	<<***extension***>> *(p. 116)* The initial number of info units that a **dds::sub::DataReader** (p. 976) can have, which are used to store **dds::sub::SampleInfo** (p. 2285).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_outstanding_reads** (p. 1090)
	<<***extension***>> *(p. 116)* The initial number of outstanding calls to read/take (or one of their variants) on the same **dds::sub::DataReader** (p. 976) for which memory has not been returned by calling **dds::sub::LoanedSamples::return_loan()** (p. 1667).

rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_outstanding_reads** (p. 1091)
	<<***extension***>> *(p. 116)* The maximum number of outstanding read/take calls (or one of their variants) on the same **dds::sub::DataReader** (p. 976) for which memory has not been returned by calling **dds::sub::Loaned↩ Samples::return_loan()** (p. 1667).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_samples_per_read** (p. 1091)
	<<***extension***>> *(p. 116)* The maximum number of data samples that the application can receive from the middleware in a single call to **dds::sub::DataReader↩ ::read** (p. 996) or **dds::sub::DataReader::take** (p. 997). If more data exists in the middleware, the application will need to issue multiple read/take calls.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**disable_fragmentation_support** (p. 1092)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) can receive fragmented samples.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragmented_samples** (p. 1092)
	<<***extension***>> *(p. 116)* The maximum number of samples for which the **dds::sub::DataReader** (p. 976) may store fragments at a given point in time.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_fragmented_samples** (p. 1093)
	<<***extension***>> *(p. 116)* The initial number of samples for which a **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragmented_samples_per_remote_writer** (p. 1094)
	<<***extension***>> *(p. 116)* The maximum number of samples per remote writer for which a **dds::sub::Data↩ Reader** (p. 976) may store fragments.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragments_per_sample** (p. 1094)
	<<***extension***>> *(p. 116)* Maximum number of fragments for a single sample.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**dynamically_allocate_fragmented_samples** (p. 1095)
	<<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.

rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_total_instances** (p. 1096)
	<<***extension***>> *(p. 116)* Maximum number of instances for which a DataReader will keep state.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_virtual_writers** (p. 1097)
	<<***extension***>> *(p. 116)* The maximum number of remote virtual writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_remote_virtual_writers** (p. 1097)
	<<***extension***>> *(p. 116)* The initial number of remote virtual writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_virtual_writers_per_instance** (p. 1098)
	<<***extension***>> *(p. 116)* The maximum number of virtual remote writers that can be associated with an instance.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_remote_virtual_writers_per_instance** (p. 1099)
	<<***extension***>> *(p. 116)* The initial number of virtual remote writers per instance.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_writers_per_sample** (p. 1099)
	<<***extension***>> *(p. 116)* The maximum number of remote writers allowed to write the same sample.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_query_condition_filters** (p. 1100)
	<<***extension***>> *(p. 116)* The maximum number of query condition filters a reader is allowed.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_app_ack_response_length** (p. 1101)
	<<***extension***>> *(p. 116)* Maximum length of application-level acknowledgment response data
rti::core::policy::DataReaderResourceLimits (p. 1081)	**keep_minimum_state_for_instances** (p. 1101)
	<<***extension***>> *(p. 116)* Whether or not keep a minimum instance state for up to **rti::core::policy::↩ DataReaderResourceLimits::max_total_instances** (p. 1096).

rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_topic_queries** (p. 1102)
	<<***extension***>> *(p. 116)* The initial number of Topic↩ Queries allocated by a **dds::sub::DataReader** (p. 976).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_topic_queries** (p. 1103)
	<<***extension***>> *(p. 116)* The maximum number of active TopicQueries that a **dds::sub::DataReader** (p. 976) can create.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**shmem_ref_transfer_mode_attached_segment_↩ allocation** (p. 1106)
	<<***extension***>> *(p. 116)* Allocation resource for the shared memory segments attached by the **dds::sub↩ ::DataReader** (p. 976).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**instance_replacement** (p. 1106)
	<<***extension***>> *(p. 116)* Sets the kind of instances allowed to be replaced for each instance state (**dds↩ ::sub::status::InstanceState** (p. 1615)) when a Data↩ Reader reaches **dds::core::policy::ResourceLimits↩ ::max_instances** (p. 2206).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**autopurge_remote_not_alive_writer_delay** (p. 1103)
	<<***extension***>> *(p. 116)* Maximum duration for which the **dds::sub::DataReader** (p. 976) will maintain information regarding a **dds::pub::DataWriter** (p. 1138) once the **dds::pub::DataWriter** (p. 1138) has become not alive.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**lazily_allocate_batch_resources** (p. 1105)
	<<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) pre-allocates resources for receiving batches of samples or lazily allocates them when needed.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_concurrent_blocking_threads** (p. 1238)
	[DEPRECATED] (p. **296)** <<***extension***>> *(p. 116)* The initial number of threads that are allowed to concurrently block on write call on the same **dds::pub::Data↩ Writer** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_concurrent_blocking_threads** (p. 1238)
	[DEPRECATED] (p. **296)** <<***extension***>> *(p. 116)* The maximum number of threads that are allowed to concurrently block on write call on the same **dds::pub::↩ DataWriter** (p. 1138).

rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_remote_reader_filters** (p. 1239)
	<<***extension***>> *(p. 116)* The maximum number of remote DataReaders for which the **dds::pub::DataWriter** (p. 1138) will perform content-based filtering.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_batches** (p. 1239)
	<<***extension***>> *(p. 116)* Represents the initial number of batches a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_batches** (p. 1240)
	<<***extension***>> *(p. 116)* Represents the maximum number of batches a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**instance_replacement** (p. 1241)
	<<***extension***>> *(p. 116)* Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**replace_empty_instances** (p. 1242)
	<<***extension***>> *(p. 116)* Whether or not to replace empty instances during instance replacement
rti::core::policy::DataWriterResourceLimits (p. 1235)	**autoregister_instances** (p. 1242)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Whether or not to automatically register new instances
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_virtual_writers** (p. 1244)
	<<***extension***>> *(p. 116)* The maximum number of virtual writers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_virtual_writers** (p. 1243)
	<<***extension***>> *(p. 116)* The initial number of virtual writers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_remote_readers** (p. 1244)
	<<***extension***>> *(p. 116)* Controls the initial memory allocation for the matched remote readers of a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_remote_readers** (p. 1244)
	<<***extension***>> *(p. 116)* The maximum number of remote readers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_app_ack_remote_readers** (p. 1245)

	<<***extension***>> *(p. 116)* The maximum number of application-level acknowledging remote readers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_active_topic_queries** (p. 1246)
	<<***extension***>> *(p. 116)* Represents the initial number of active topic queries a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_active_topic_queries** (p. 1246)
	<<***extension***>> *(p. 116)* Represents the maximum number of active topic queries a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**writer_loaned_sample_allocation** (p. 1247)
	<<***extension***>> *(p. 116)* Represents the allocation settings of loaned samples managed by a **dds::pub↩ ::DataWriter** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initialize_writer_loaned_sample** (p. 1248)
	<<***extension***>> *(p. 116)* Whether or not to initialize loaned samples returned by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterTransferMode (p. 1253)	**shmem_ref_settings** (p. 1254)
	<<***extension***>> *(p. 116)* Settings related to transferring data using shared memory references.
rti::core::policy::DataReaderProtocol (p. 1059)	**virtual_guid** (p. 1060)
	<<***extension***>> *(p. 116)* The virtual GUID (Global Unique Identifier).
rti::core::policy::DataReaderProtocol (p. 1059)	**rtps_object_id** (p. 1061)
	<<***extension***>> *(p. 116)* The RTPS Object ID.
rti::core::policy::DataReaderProtocol (p. 1059)	**rtps_reliable_reader** (p. 1064)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **dds::sub::DataReader** (p. 976). This parameter only has effect if the reader is configured with **dds::core::policy::ReliabilityKind_def::↩ RELIABLE** (p. 2157) **dds::core::policy::Reliability↩ Kind_def** (p. 2156).
rti::core::policy::DataReaderProtocol (p. 1059)	**expects_inline_qos** (p. 1062)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Specifies whether this DataReader expects inline QoS with every sample.
rti::core::policy::DataReaderProtocol (p. 1059)	**disable_positive_acks** (p. 1062)

	<<***extension***>> *(p. 116)* Whether the reader sends positive acknowledgements to writers.
rti::core::policy::DataReaderProtocol (p. 1059)	**propagate_dispose_of_unregistered_instances** (p. 1063)
	<<***extension***>> *(p. 116)* Indicates whether or not an instance can move to the **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) state without being in the **dds::sub::status::InstanceState::alive()** (p. 1617) state.
rti::core::policy::DataReaderProtocol (p. 1059)	**propagate_unregister_of_disposed_instances** (p. 1064)
	<<***extension***>> *(p. 116)* Indicates whether or not an instance can move to the **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) state directly from the **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617).
rti::core::policy::DataWriterProtocol (p. 1210)	**virtual_guid** (p. 1212)
	<<***extension***>> *(p. 116)* The virtual GUID (Global Unique Identifier).
rti::core::policy::DataWriterProtocol (p. 1210)	**rtps_object_id** (p. 1212)
	<<***extension***>> *(p. 116)* The RTPS Object ID.
rti::core::policy::DataWriterProtocol (p. 1210)	**push_on_write** (p. 1213)
	<<***extension***>> *(p. 116)* Whether to push sample out when write is called.
rti::core::policy::DataWriterProtocol (p. 1210)	**disable_positive_acks** (p. 1214)
	<<***extension***>> *(p. 116)* Controls whether or not the writer expects positive acknowledgements from matching readers.
rti::core::policy::DataWriterProtocol (p. 1210)	**disable_inline_keyhash** (p. 1214)
	<<***extension***>> *(p. 116)* Controls whether or not a keyhash is propagated on the wire with each sample.
rti::core::policy::DataWriterProtocol (p. 1210)	**serialize_key_with_dispose** (p. 1215)
	<<***extension***>> *(p. 116)* Controls whether or not the serialized key is propagated on the wire with dispose samples.
rti::core::policy::DataWriterProtocol (p. 1210)	**propagate_app_ack_with_no_response** (p. 1216)
	<<***extension***>> *(p. 116)* Controls whether or not a **dds::pub::DataWriter** (p. 1138) receives **dds::pub::DataWriterListener::on_application_acknowledgment** (p. 1207) notifications with an empty or invalid response.

rti::core::policy::DataWriterProtocol (p. 1210)	**rtps_reliable_writer** (p. 1217)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **dds::pub::DataWriter** (p. 1138). This parameter only has effect if both the writer and the matching reader are configured with **dds::core::policy::↩ ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core↩ ::policy::ReliabilityKind_def** (p. 2156).
rti::core::policy::DataWriterProtocol (p. 1210)	**initial_virtual_sequence_number** (p. 1218)
	<<***extension***>> *(p. 116)* Determines, the initial virtual sequence number for this DataWriter.
rti::core::policy::SystemResourceLimits (p. 2443)	**max_objects_per_thread** (p. 2445)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* The maximum number of objects that can be stored per thread for a DomainParticipantFactory.
rti::core::policy::SystemResourceLimits (p. 2443)	**initial_objects_per_thread** (p. 2445)
	<<***extension***>> *(p. 116)* The number of objects per thread for a DomainParticipantFactory for which infrastructure will initially be allocated.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_writer_allocation** (p. 1398)
	<<***extension***>> *(p. 116)* Allocation settings applied to local DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_reader_allocation** (p. 1398)
	<<***extension***>> *(p. 116)* Allocation settings applied to local DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_publisher_allocation** (p. 1399)
	<<***extension***>> *(p. 116)* Allocation settings applied to local Publisher.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_subscriber_allocation** (p. 1399)
	<<***extension***>> *(p. 116)* Allocation settings applied to local Subscriber.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_topic_allocation** (p. 1399)
	<<***extension***>> *(p. 116)* Allocation settings applied to local Topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_writer_allocation** (p. 1400)

	<<***extension***>> *(p. 116)* Allocation settings applied to remote DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_reader_allocation** (p. 1401)
	<<***extension***>> *(p. 116)* Allocation settings applied to remote DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_participant_allocation** (p. 1401)
	<<***extension***>> *(p. 116)* Allocation settings applied to remote DomainParticipants.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_writer_reader_pair_allocation** (p. 1402)
	<<***extension***>> *(p. 116)* Allocation settings applied to matching local writer and remote/local reader pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_reader_writer_pair_allocation** (p. 1402)
	<<***extension***>> *(p. 116)* Allocation settings applied to matching local reader and remote/local writer pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**ignored_entity_allocation** (p. 1403)
	<<***extension***>> *(p. 116)* Allocation settings applied to ignored entities.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filtered_topic_allocation** (p. 1403)
	<<***extension***>> *(p. 116)* Allocation settings applied to content filtered topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filter_allocation** (p. 1404)
	<<***extension***>> *(p. 116)* Allocation settings applied to content filter.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**read_condition_allocation** (p. 1404)
	<<***extension***>> *(p. 116)* Allocation settings applied to read condition pool.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**query_condition_allocation** (p. 1405)
	<<***extension***>> *(p. 116)* Allocation settings applied to query condition pool.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**outstanding_asynchronous_sample_allocation** (p. 1405)

	<<***extension***>> *(p. 116)* Allocation settings applied to the maximum number of samples (from all **dds::pub::↩ DataWriter** (p. 1138)) waiting to be asynchronously written.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**flow_controller_allocation** (p. 1406)
	<<***extension***>> *(p. 116)* Allocation settings applied to flow controllers.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_writer_hash_buckets** (p. 1406)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Hash_Buckets settings applied to local DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_reader_hash_buckets** (p. 1407)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Number of hash buckets for local DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_publisher_hash_buckets** (p. 1407)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Number of hash buckets for local Publisher.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_subscriber_hash_buckets** (p. 1408)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Number of hash buckets for local Subscriber.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_topic_hash_buckets** (p. 1408)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Number of hash buckets for local Topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_writer_hash_buckets** (p. 1409)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Number of hash buckets for remote DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_reader_hash_buckets** (p. 1409)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Number of hash buckets for remote DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_participant_hash_buckets** (p. 1410)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Number of hash buckets for remote DomainParticipants.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_writer_reader_pair_hash_buckets** (p. 1410)

	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for matching local writer and remote/local reader pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_reader_writer_pair_hash_buckets** (p. 1411)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for matching local reader and remote/local writer pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**ignored_entity_hash_buckets** (p. 1411)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for ignored entities.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filtered_topic_hash_buckets** (p. 1412)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for content filtered topics.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filter_hash_buckets** (p. 1412)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for content filters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**flow_controller_hash_buckets** (p. 1413)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for flow controllers.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_gather_destinations** (p. 1413)
	<<***extension***>> *(p. 116)* Maximum number of destinations per RTI Connext send.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**participant_user_data_max_length** (p. 1414)
	<<***extension***>> *(p. 116)* Maximum length of user data in **dds::domain::qos::DomainParticipantQos** (p. 1384) and **dds::topic::ParticipantBuiltinTopicData** (p. 1908)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**topic_data_max_length** (p. 1414)
	<<***extension***>> *(p. 116)* Maximum length of topic data in **dds::topic::qos::TopicQos** (p. 2509), **dds::topic::TopicBuiltinTopicData** (p. 2493), **dds::topic::PublicationBuiltinTopicData** (p. 1977) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**publisher_group_data_max_length** (p. 1415)
	<<***extension***>> *(p. 116)* Maximum length of group data in **dds::pub::qos::PublisherQos** (p. 2007) and **dds::topic::PublicationBuiltinTopicData** (p. 1977)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**subscriber_group_data_max_length** (p. 1415)
	<<***extension***>> *(p. 116)* Maximum length of group data in **dds::sub::qos::SubscriberQos** (p. 2423) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_user_data_max_length** (p. 1416)
	<<***extension***>> *(p. 116)* Maximum length of user data in **dds::pub::qos::DataWriterQos** (p. 1226) and **dds::topic::PublicationBuiltinTopicData** (p. 1977)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_user_data_max_length** (p. 1416)
	<<***extension***>> *(p. 116)* Maximum length of user data in **dds::sub::qos::DataReaderQos** (p. 1072) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_partitions** (p. 1417)
	<<***extension***>> *(p. 116)* Maximum number of partition name strings allowable in a **dds::core::policy::Partition** (p. 1923).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_partition_cumulative_characters** (p. 1417)
	<<***extension***>> *(p. 116)* Maximum number of combined characters allowable in all partition names in a **dds::core::policy::Partition** (p. 1923).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_code_max_serialized_length** (p. 1418)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Maximum size of serialized string for type code.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_object_max_serialized_length** (p. 1419)
	<<***extension***>> *(p. 116)* The maximum length, in bytes, that the buffer to serialize a TypeObject v1 can consume.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_object_max_deserialized_length** (p. 1420)
	<<***extension***>> *(p. 116)* The maximum number of bytes that a deserialized TypeObject v1 can consume.

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**deserialized_type_object_dynamic_allocation_threshold** (p. 1420)
	<<***extension***>> *(p. 116)* A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject v1.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**serialized_type_object_dynamic_allocation_threshold** (p. 1421)
	<<***extension***>> *(p. 116)* A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject v1.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**contentfilter_property_max_length** (p. 1422)
	<<***extension***>> *(p. 116)* This field is the maximum length of all data related to a Content-filtered topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**channel_seq_max_length** (p. 1423)
	<<***extension***>> *(p. 116)* Maximum number of channels that can be specified in **rti::core::policy::MultiChannel** (p. 1749) for MultiChannel DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**channel_filter_expression_max_length** (p. 1424)
	<<***extension***>> *(p. 116)* Maximum length of a channel **rti::core::ChannelSettings::filter_expression** (p. 923) in a MultiChannel DataWriter.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**participant_property_list_max_length** (p. 1424)
	<<***extension***>> *(p. 116)* Maximum number of properties associated with the **dds::domain::DomainParticipant** (p. 1326).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**participant_property_string_max_length** (p. 1425)
	<<***extension***>> *(p. 116)* Maximum string length of the properties associated with the **dds::domain::DomainParticipant** (p. 1326).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_property_list_max_length** (p. 1425)
	<<***extension***>> *(p. 116)* Maximum number of properties associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_property_string_max_length** (p. 1426)

	<<***extension***>> *(p. 116)* Maximum string length of the properties associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_property_list_max_length** (p. 1426)
	<<***extension***>> *(p. 116)* Maximum number of properties associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_property_string_max_length** (p. 1427)
	<<***extension***>> *(p. 116)* Maximum string length of the properties associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_endpoint_groups** (p. 1427)
	<<***extension***>> *(p. 116)* Maximum number of **rti::core::EndpointGroup** (p. 1515) allowable in a **rti::core::policy::Availability** (p. 868).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_endpoint_group_cumulative_characters** (p. 1428)
	<<***extension***>> *(p. 116)* Maximum number of combined role_name characters allowed in all **rti::core::EndpointGroup** (p. 1515) in a **rti::core::policy::Availability** (p. 868).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**transport_info_list_max_length** (p. 1428)
	<<***extension***>> *(p. 116)* Maximum number of installed transports to send and receive information about in **dds::topic::ParticipantBuiltinTopicData::transport_info** (p. 1914).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**ignored_entity_replacement_kind** (p. 1429)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **rti::core::policy::DomainParticipantResourceLimits::ignored_entity_allocation** (p. 1403) are reached.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_topic_query_allocation** (p. 1429)
	<<***extension***>> *(p. 116)* Allocation settings applied to remote TopicQueries.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_topic_query_hash_buckets** (p. 1430)

	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote TopicQueries.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_data_tag_list_max_length** (p. 1431)
	<<***extension***>> *(p. 116)* Maximum number of data tags associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_data_tag_string_max_length** (p. 1431)
	<<***extension***>> *(p. 116)* Maximum string length of the data tags associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_data_tag_list_max_length** (p. 1432)
	<<***extension***>> *(p. 116)* Maximum number of data tags associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_data_tag_string_max_length** (p. 1432)
	<<***extension***>> *(p. 116)* Maximum string length of the data tags associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**shmem_ref_transfer_mode_max_segments** (p. 1433)
	<<***extension***>> *(p. 116)* Maximum number of segments created by all DataWriters belonging to a **dds::domain::DomainParticipant** (p. 1326).
rti::core::policy::Event (p. 1541)	**thread** (p. 1542)
	<<***extension***>> *(p. 116)* Event thread QoS.
rti::core::policy::Event (p. 1541)	**initial_count** (p. 1543)
	<<***extension***>> *(p. 116)* The initial number of events.
rti::core::policy::Event (p. 1541)	**max_count** (p. 1543)
	<<***extension***>> *(p. 116)* The maximum number of events.
rti::core::policy::Database (p. 969)	**thread** (p. 971)
	<<***extension***>> *(p. 116)* Database thread settings.
rti::core::policy::Database (p. 969)	**shutdown_timeout** (p. 972)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum wait time during a shutdown.
rti::core::policy::Database (p. 969)	**cleanup_period** (p. 972)

	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The period at which the database thread wakes up to remove deleted records.
rti::core::policy::Database (p. 969)	**shutdown_cleanup_period** (p. 973)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The clean-up period used during database shut-down.
rti::core::policy::Database (p. 969)	**initial_records** (p. 974)
	<<***extension***>> *(p. 116)* The initial number of total records.
rti::core::policy::Database (p. 969)	**max_skiplist_level** (p. 974)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum level of the skiplist.
rti::core::policy::Database (p. 969)	**initial_weak_references** (p. 975)
	<<***extension***>> *(p. 116)* The initial number of weak references.
rti::core::policy::Database (p. 969)	**max_weak_references** (p. 974)
	<<***extension***>> *(p. 116)* The maximum number of weak references.
rti::core::policy::ReceiverPool (p. 2144)	**thread** (p. 2146)
	<<***extension***>> *(p. 116)* Receiver pool thread(s).
rti::core::policy::ReceiverPool (p. 2144)	**buffer_size** (p. 2147)
	<<***extension***>> *(p. 116)* The receive buffer size in bytes.
rti::core::policy::ReceiverPool (p. 2144)	**buffer_alignment** (p. 2148)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The receive buffer alignment.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_liveliness_lease_duration** (p. 1279)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The liveliness lease duration for the participant.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_liveliness_assert_period** (p. 1279)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The period to assert liveliness for the participant.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_announcement_period** (p. 1280)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
rti::core::policy::DiscoveryConfig (p. 1271)	**remote_participant_purge_kind** (p. 1281)

	[DEPRECATED] (p. **296**) *<<**extension**>> (p. 116)* The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
rti::core::policy::DiscoveryConfig (p. 1271)	**max_liveliness_loss_detection_period** (p. 1281)
	[DEPRECATED] (p. **296**) *<<**extension**>> (p. 116)* The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.
rti::core::policy::DiscoveryConfig (p. 1271)	**initial_participant_announcements** (p. 1282)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The number of initial announcements sent when a participant is first enabled.
rti::core::policy::DiscoveryConfig (p. 1271)	**new_remote_participant_announcements** (p. 1283)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The number of participant announcements sent when a remote participant is newly discovered.
rti::core::policy::DiscoveryConfig (p. 1271)	**min_initial_participant_announcement_period** (p. 1283)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
rti::core::policy::DiscoveryConfig (p. 1271)	**max_initial_participant_announcement_period** (p. 1284)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_reader_resource_limits** (p. 1285)
	*<<**extension**>> (p. 116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_reader** (p. 1286)
	*<<**extension**>> (p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_reader_resource_limits** (p. 1287)
	*<<**extension**>> (p. 116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_reader** (p. 1288)
	*<<**extension**>> (p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.

rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_reader_resource_limits** (p. 1289)
	<<***extension***>> *(p. 116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_writer** (p. 1296)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core::policy::ReliabilityKind_def** (p. 2156).
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_writer_publish_mode** (p. 1298)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in publication writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_writer_publish_mode** (p. 1298)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in subscription writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**asynchronous_publisher** (p. 1299)
	<<***extension***>> *(p. 116)* Asynchronous publishing settings for the discovery **dds::pub::Publisher** (p. 1993) and all entities that are created by it.
rti::core::policy::DiscoveryConfig (p. 1271)	**default_domain_announcement_period** (p. 1300)
	<<***extension***>> *(p. 116)* The period to announce a participant to the default domain 0.
rti::core::policy::DiscoveryConfig (p. 1271)	**ignore_default_domain_announcements** (p. 1301)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_reader_reliability_kind** (p. 1295)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Reliability policy for a built-in participant message reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_reader** (p. 1295)
	<<***extension***>> *(p. 116)* RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **rti::core::policy::DiscoveryConfig::participant_message_reader_reliability_kind** (p. 1295) is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157).
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_writer** (p. 1289)

	<<***extension***>> *(*p. *116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_writer_data_lifecycle** (p. 1291)
	<<***extension***>> *(*p. *116)* Writer data lifecycle settings for a built-in publication writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_writer** (p. 1291)
	<<***extension***>> *(*p. *116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_writer_data_lifecycle** (p. 1293)
	<<***extension***>> *(*p. *116)* Writer data lifecycle settings for a built-in subscription writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**builtin_discovery_plugins** (p. 1293)
	<<***extension***>> *(*p. *116)* Mask of built-in discovery plugin kinds.
rti::core::policy::DiscoveryConfig (p. 1271)	**enabled_builtin_channels** (p. 1294)
	<<***extension***>> *(*p. *116)* The mask specifying which built-in channels should be enabled.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_writer** (p. 1301)
	<<***extension***>> *(*p. *116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **rti::topic::ServiceRequest** (p. 2358) writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_writer_data_lifecycle** (p. 1303)
	<<***extension***>> *(*p. *116)* Writer data lifecycle settings for a built-in **rti::topic::ServiceRequest** (p. 2358) writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_writer_publish_mode** (p. 1303)
	<<***extension***>> *(*p. *116)* Publish mode policy for the built-in service request writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_reader** (p. 1304)
	<<***extension***>> *(*p. *116)* RTPS reliable reader protocol-related configuration settings for a built-in **rti::topic::ServiceRequest** (p. 2358) reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**locator_reachability_lease_duration** (p. 1306)
	<<***extension***>> *(*p. *116)* The time period after which other DomainParticipants can consider one of their locators as "unreachable" if they do not receive a REACHABILITY PING from this DomainParticipant.

rti::core::policy::DiscoveryConfig (p. 1271)	**locator_reachability_assert_period** (p. 1305)
	<<***extension***>> *(p. 116)* Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.
rti::core::policy::DiscoveryConfig (p. 1271)	**locator_reachability_change_detection_period** (p. 1306)
	<<***extension***>> *(p. 116)* Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_writer** (p. 1307)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_writer_publish_mode** (p. 1309)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in secure volatile writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_reader** (p. 1309)
	<<***extension***>> *(p. 116)* RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**endpoint_type_object_lb_serialization_threshold** (p. 1310)
	<<***extension***>> *(p. 116)* Controls when/if the Type↩ Object v1 is compressed when propagated in Simple Endpoint Discovery.
rti::core::policy::DiscoveryConfig (p. 1271)	**dns_tracker_polling_period** (p. 1311)
	<<***extension***>> *(p. 116)* Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_reader** (p. 1312)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_reader_resource_limits** (p. 1313)
	<<***extension***>> *(p. 116)* Resource limits for the built-in topic participant configuration reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_writer_publish_mode** (p. 1316)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in participant configuration writer.

rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_writer** (p. 1314)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_writer_data_lifecycle** (p. 1316)
	<<***extension***>> *(p. 116)* Writer data lifecycle settings for a built-in participant configuration writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**request_types_filter** (p. 1317)
	<<***extension***>> *(p. 116)* Topic name pattern for requesting all types via Type Lookup Service.
rti::core::RtpsWellKnownPorts (p. 2258)	**port_base** (p. 2262)
	<<***extension***>> *(p. 116)* The base port offset.
rti::core::RtpsWellKnownPorts (p. 2258)	**domain_id_gain** (p. 2262)
	<<***extension***>> *(p. 116)* Tunable domain gain parameter.
rti::core::RtpsWellKnownPorts (p. 2258)	**participant_id_gain** (p. 2263)
	<<***extension***>> *(p. 116)* Tunable participant gain parameter.
rti::core::RtpsWellKnownPorts (p. 2258)	**builtin_multicast_port_offset** (p. 2263)
	<<***extension***>> *(p. 116)* Additional offset for **metatraffic** multicast port.
rti::core::RtpsWellKnownPorts (p. 2258)	**builtin_unicast_port_offset** (p. 2264)
	<<***extension***>> *(p. 116)* Additional offset for **metatraffic** unicast port.
rti::core::RtpsWellKnownPorts (p. 2258)	**user_multicast_port_offset** (p. 2264)
	<<***extension***>> *(p. 116)* Additional offset for **usertraffic** multicast port.
rti::core::RtpsWellKnownPorts (p. 2258)	**user_unicast_port_offset** (p. 2265)
	<<***extension***>> *(p. 116)* Additional offset for **usertraffic** unicast port.
rti::core::RtpsReliableReaderProtocol (p. 2221)	**min_heartbeat_response_delay** (p. 2223)
	<<***extension***>> *(p. 116)* The minimum delay to respond to a heartbeat.
rti::core::RtpsReliableReaderProtocol (p. 2221)	**max_heartbeat_response_delay** (p. 2224)
	<<***extension***>> *(p. 116)* The maximum delay to respond to a heartbeat.

rti::core::RtpsReliableReaderProtocol (p. 2221)	**heartbeat_suppression_duration** (p. 2224)
	<<***extension***>> *(p. 116)* The duration a reader ignores consecutively received heartbeats
rti::core::RtpsReliableReaderProtocol (p. 2221)	**nack_period** (p. 2225)
	<<***extension***>> *(p. 116)* The period at which to send NACKs
rti::core::RtpsReliableReaderProtocol (p. 2221)	**receive_window_size** (p. 2226)
	<<***extension***>> *(p. 116)* The number of received out-of-order samples a reader can keep at a time
rti::core::RtpsReliableReaderProtocol (p. 2221)	**round_trip_time** (p. 2226)
	<<***extension***>> *(p. 116)* The duration from sending a NACK to receiving a repair of a sample
rti::core::RtpsReliableReaderProtocol (p. 2221)	**app_ack_period** (p. 2227)
	<<***extension***>> *(p. 116)* The period at which application-level acknowledgment messages are sent
rti::core::RtpsReliableReaderProtocol (p. 2221)	**min_app_ack_response_keep_duration** (p. 2227)
	<<***extension***>> *(p. 116)* Minimum duration for which application-level acknowledgment response data is kept
rti::core::RtpsReliableReaderProtocol (p. 2221)	**samples_per_app_ack** (p. 2228)
	<<***extension***>> *(p. 116)* The minimum number of samples acknowledged by one application-level acknowledgment message
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**low_watermark** (p. 2232)
	<<***extension***>> *(p. 116)* When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385) is considered to have changed.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**high_watermark** (p. 2233)
	<<***extension***>> *(p. 116)* When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, the **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385) is considered to have changed.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**heartbeat_period** (p. 2234)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The period at which to send heartbeats.

rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**fast_heartbeat_period** (p. 2235)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**late_joiner_heartbeat_period** (p. 2236)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**virtual_heartbeat_period** (p. 2237)
	<<***extension***>> *(p. 116)* The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**samples_per_virtual_heartbeat** (p. 2238)
	<<***extension***>> *(p. 116)* The number of samples that a reliable writer has to publish before sending a virtual heartbeat.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_heartbeat_retries** (p. 2239)
	<<***extension***>> *(p. 116)* The maximum number of *periodic* heartbeat retries before marking a remote reader as inactive.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**inactivate_nonprogressing_readers** (p. 2239)
	<<***extension***>> *(p. 116)* Whether to treat remote readers as inactive when their NACKs do not progress
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**heartbeats_per_max_samples** (p. 2240)
	<<***extension***>> *(p. 116)* The number of piggyback heartbeats sent per max send window.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**min_nack_response_delay** (p. 2242)
	<<***extension***>> *(p. 116)* The minimum delay to respond to a NACK or NACK_FRAG.

rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_nack_response_delay** (p. 2242)
	<<***extension***>> *(p. 116)* The maximum delay to respond to a NACK or NACK_FRAG.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**nack_suppression_duration** (p. 2243)
	<<***extension***>> *(p. 116)* The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_bytes_per_nack_response** (p. 2244)
	<<***extension***>> *(p. 116)* The maximum total message size when resending rejected samples or data fragments.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_enable_adaptive_sample_↩ keep_duration** (p. 2247)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Enables dynamic adjustment of sample keep duration in response to congestion.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_min_sample_keep_duration** (p. 2245)
	<<***extension***>> *(p. 116)* The minimum duration a sample is queued for ACK-disabled readers.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_max_sample_keep_duration** (p. 2246)
	<<***extension***>> *(p. 116)* The maximum duration a sample is queued for ACK-disabled readers.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_decrease_sample_keep_↩ duration_factor** (p. 2248)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Controls rate of contraction of dynamic sample keep duration.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_increase_sample_keep_↩ duration_factor** (p. 2248)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Controls rate of growth of dynamic sample keep duration.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**min_send_window_size** (p. 2249)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Minimum size of send window of unacknowledged samples

rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_send_window_size** (p. 2250)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Maximum size of send window of unacknowledged samples
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**send_window_update_period** (p. 2251)
	<<***extension***>> *(p. 116)* Period in which send window may be dynamically changed
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**send_window_increase_factor** (p. 2252)
	<<***extension***>> *(p. 116)* Increases send window size by this percentage when reacting dynamically to network conditions
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**send_window_decrease_factor** (p. 2253)
	<<***extension***>> *(p. 116)* Decreases send window size by this percentage when reacting dynamically to network conditions
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**enable_multicast_periodic_heartbeat** (p. 2254)
	<<***extension***>> *(p. 116)* Whether periodic heartbeat messages are sent over multicast
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**multicast_resend_threshold** (p. 2254)
	<<***extension***>> *(p. 116)* The minimum number of requesting readers needed to trigger a multicast resend
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_repair_piggyback_heartbeat** (p. 2255)
	<<***extension***>> *(p. 116)* Prevents piggyback heartbeats from being sent with repair samples
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_samples** (p. 909)
	<<***extension***>> *(p. 116)* Initial number of samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_samples** (p. 909)
	<<***extension***>> *(p. 116)* Maximum number of samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_infos** (p. 910)
	<<***extension***>> *(p. 116)* Initial number of sample infos.

rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_infos** (p. 911)
	<<***extension***>> *(p. 116)* Maximum number of sample infos.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_outstanding_reads** (p. 911)
	<<***extension***>> *(p. 116)* The initial number of outstanding reads that have not called finish yet on the same built-in topic **dds::sub::DataReader** (p. 976).
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_outstanding_reads** (p. 912)
	<<***extension***>> *(p. 116)* The maximum number of outstanding reads that have not called finish yet on the same built-in topic **dds::sub::DataReader** (p. 976).
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_samples_per_read** (p. 912)
	<<***extension***>> *(p. 116)* Maximum number of samples that can be read/taken on a same built-in topic **dds::sub::DataReader** (p. 976).
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**disable_fragmentation_support** (p. 913)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Determines whether the built-in topic **dds::sub::Data↩ Reader** (p. 976) can receive fragmented samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragmented_samples** (p. 913)
	<<***extension***>> *(p. 116)* The maximum number of samples for which the built-in topic **dds::sub::Data↩ Reader** (p. 976) may store fragments at a given point in time.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_fragmented_samples** (p. 914)
	<<***extension***>> *(p. 116)* The initial number of samples for which a built-in topic **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragmented_samples_per_remote_writer** (p. 915)
	<<***extension***>> *(p. 116)* The maximum number of samples per remote writer for which a built-in topic **dds↩ ::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragments_per_sample** (p. 916)

	<<***extension***>> *(*p. *116)* Maximum number of fragments for a single sample.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**dynamically_allocate_fragmented_samples** (p. 916)
	<<***extension***>> *(*p. *116)* Determines whether the built-in topic **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.
rti::core::TransportUnicastSettings (p. 2559)	**transports** (p. 2561)
	<<***extension***>> *(*p. *116)* A sequence of transport aliases that specifies the unicast interfaces on which to receive *unicast* traffic for the entity.
rti::core::TransportUnicastSettings (p. 2559)	**receive_port** (p. 2561)
	<<***extension***>> *(*p. *116)* The unicast port on which the entity can receive data.
rti::core::TransportMulticastSettings (p. 2549)	**transports** (p. 2550)
	<<***extension***>> *(*p. *116)* A sequence of transport aliases that specifies the transports on which to receive *multicast* traffic for the entity.
rti::core::TransportMulticastSettings (p. 2549)	**receive_address** (p. 2551)
	<<***extension***>> *(*p. *116)* The multicast group address on which the entity can receive data.
rti::core::TransportMulticastSettings (p. 2549)	**receive_port** (p. 2551)
	<<***extension***>> *(*p. *116)* The multicast port on which the entity can receive data.
rti::core::policy::MulticastMapping (p. 1747)	**addresses** (p. 1748)
	<<***extension***>> *(*p. *116)* A string containing a comma-separated list of IP addresses or IP address ranges to be used to receive *multicast* traffic for the entity with a topic that matches the **rti::core::policy::Multicast↩ Mapping::topic_expression** (p. 1748).
rti::core::policy::MulticastMapping (p. 1747)	**topic_expression** (p. 1748)
	<<***extension***>> *(*p. *116)* A regular expression that will be used to map topic names to corresponding multicast receive addresses.
rti::core::policy::MulticastMapping (p. 1747)	**mapping_function** (p. 1749)
	<<***extension***>> *(*p. *116)* Specifies a function that will define the mapping between a topic name and a specific multicast address from a list of addresses.
rti::core::policy::Property (p. 1969)	**value** (p. 1973)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Sequence of properties.

rti::core::policy::EntityName (p. 1530)	**name** (p. 1531)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The name of the entity.
rti::core::policy::EntityName (p. 1530)	**role_name** (p. 1532)
	*<<**extension**>> (p. 116)* The entity role name.
rti::core::policy::Service (p. 2350)	**kind** (p. 2354)
	*<<**extension**>> (p. 116)* The kind of service.
rti::core::policy::Batch (p. 880)	**enable** (p. 882)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Specifies whether or not batching is enabled.
rti::core::policy::Batch (p. 880)	**max_data_bytes** (p. 882)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The maximum cumulative length of all serialized samples in a batch.
rti::core::policy::Batch (p. 880)	**max_samples** (p. 883)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The maximum number of samples in a batch.
rti::core::policy::Batch (p. 880)	**max_flush_delay** (p. 884)
	*<<**extension**>> (p. 116)* The maximum flush delay.
rti::core::policy::Batch (p. 880)	**source_timestamp_resolution** (p. 885)
	*<<**extension**>> (p. 116)* Batch source timestamp resolution.
rti::core::policy::Batch (p. 880)	**thread_safe_write** (p. 886)
	*<<**extension**>> (p. 116)* Determines whether or not the write operation is thread safe.
rti::core::QosProviderParams (p. 2049)	**string_profile** (p. 2050)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sequence of strings containing a XML document to load.
rti::core::QosProviderParams (p. 2049)	**url_profile** (p. 2050)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sequence of **URL groups** (p. 304) containing a set of XML documents to load.
rti::core::QosProviderParams (p. 2049)	**ignore_user_profile** (p. 2051)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Ignores the file USER_QOS_PROFILES.xml in the current working directory.

rti::core::QosProviderParams (p. 2049)	**ignore_environment_profile** (p. 2052)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) Ignores the value of the **NDDS_QOS_PROFILES** (p. 304).
rti::core::QosProviderParams (p. 2049)	**ignore_resource_profile** (p. 2052)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) Ignores the file NDDS_QOS_PROFILES.xml.
rti::config::Logger (p. 1683)	**verbosity** (p. 1685)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) The verbosities at which RTI Connext diagnostic information is logged.
rti::config::Logger (p. 1683)	**category** (p. 1685)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) Categories of logged messages.
rti::config::Logger (p. 1683)	**print_format** (p. 1688)
	*<<**extension**>>* (p. *116*) The format used to output RTI Connext diagnostic information.
rti::config::Logger (p. 1683)	**output_file** (p. 1686)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) Specifies the file to which log messages will be redirected to.
rti::config::Logger (p. 1683)	**output_file_suffix** (p. 1687)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) Sets the file suffix when logging to a set of files
rti::config::Logger (p. 1683)	**max_bytes_per_file** (p. 1687)
	*<<**extension**>>* (p. *116*) Specifies the maximum number of bytes a single file can contain
rti::config::Logger (p. 1683)	**max_files** (p. 1687)
	*<<**extension**>>* (p. *116*) Specifies the maximum number of files to create before overwritting the previous ones.
rti::core::MonitoringEventDistributionSettings (p. 1722)	**concurrency_level** (p. 1723)
	*<<**extension**>>* (p. *116*) Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
rti::core::MonitoringEventDistributionSettings (p. 1722)	**datawriter_qos_profile_name** (p. 1724)
	*<<**extension**>>* (p. *116*) The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes event metrics.

rti::core::MonitoringEventDistributionSettings (p. 1722)	**thread** (p. 1726)
	<<***extension***>> *(p. 116)* The settings of the event metric thread.
rti::core::MonitoringEventDistributionSettings (p. 1722)	**publication_period** (p. 1726)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**datawriter_qos_profile_name** (p. 1742)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes periodic metrics.
rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**thread** (p. 1743)
	<<***extension***>> *(p. 116)* The settings of the periodic metric thread.
rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**polling_period** (p. 1744)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Period at which the periodic metric thread polls and publishes the periodic metrics.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**concurrency_level** (p. 1729)
	<<***extension***>> *(p. 116)* Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**max_historical_logs** (p. 1729)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**datawriter_qos_profile_name** (p. 1730)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes log messages.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**thread** (p. 1731)
	<<***extension***>> *(p. 116)* The settings of the logging thread.

rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**publication_period** (p. 1732)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Period at which the logging thread publishes log messages.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**enable** (p. 1713)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Enables the use of a dedicated **dds::domain::DomainParticipant** (p. 1326) to distribute the RTI Connext application telemetry data.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**domain_id** (p. 1713)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The domain ID used in the creation of RTI Monitoring Library 2.0 **dds::domain::DomainParticipant** (p. 1326).
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**participant_qos_profile_name** (p. 1714)
	*<<**extension**>> (p. 116)* The fully qualified name of the profile used to configure the **dds::domain::DomainParticipant** (p. 1326) that will be used to distribute telemetry data.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**collector_initial_peers** (p. 1715)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **rti::core::MonitoringDistributionSettings::dedicated_participant** (p. 1719).
rti::core::MonitoringDistributionSettings (p. 1716)	**dedicated_participant** (p. 1719)
	*<<**extension**>> (p. 116)* Configures the use of a dedicated **dds::domain::DomainParticipant** (p. 1326) to distribute the RTI Connext application telemetry data.
rti::core::MonitoringDistributionSettings (p. 1716)	**publisher_qos_profile_name** (p. 1718)
	*<<**extension**>> (p. 116)* The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.
rti::core::MonitoringDistributionSettings (p. 1716)	**event_settings** (p. 1720)
	*<<**extension**>> (p. 116)* Configures the distribution of event metrics.
rti::core::MonitoringDistributionSettings (p. 1716)	**periodic_settings** (p. 1720)
	*<<**extension**>> (p. 116)* Configures the distribution of periodic metrics.

rti::core::MonitoringDistributionSettings (p. 1716)	**logging_settings** (p. 1721)
	<<***extension***>> *(p. 116)* Configures the distribution of logging messages.
rti::core::MonitoringMetricSelection (p. 1736)	**resource_selection** (p. 1737)
	<<***extension***>> *(p. 116)* An expression pattern that selects a subset of resources by matching the pattern to the resource names. **rti::core::MonitoringMetric↩ Selection::enabled_metrics_selection** (p. 1738) and **rti::core::MonitoringMetricSelection::disabled_↩ metrics_selection** (p. 1739) are applied to this subset of resources.
rti::core::MonitoringMetricSelection (p. 1736)	**enabled_metrics_selection** (p. 1738)
	<<***extension***>> *(p. 116)* A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by **rti::core::Monitoring↩ MetricSelection::resource_selection** (p. 1737).
rti::core::MonitoringMetricSelection (p. 1736)	**disabled_metrics_selection** (p. 1739)
	<<***extension***>> *(p. 116)* A sequence of POSIX fnmatch patterns that mach the names of the metrics that should not be collected and distributed for the observable resources selected by **rti::core::Monitoring↩ MetricSelection::resource_selection** (p. 1737).
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**middleware_forwarding_level** (p. 1734)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Log messages with **rti::config::LogFacility::middleware** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**security_event_forwarding_level** (p. 1734)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Log messages with **rti::config::LogFacility::security_↩ event** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**service_forwarding_level** (p. 1735)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Log messages with **rti::config::LogFacility::service** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**user_forwarding_level** (p. 1736)

	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Log messages with **rti::config::LogFacility::user** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringTelemetryData (p. 1744)	**metrics** (p. 1745)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Sequence of **rti::core::MonitoringMetricSelection** (p. 1736) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.
rti::core::MonitoringTelemetryData (p. 1744)	**logs** (p. 1746)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* **rti::core::MonitoringLoggingForwardingSettings** (p. 1733) containing the **rti::config::SyslogVerbosity** (p. 458) levels that will be forwarded for the different **rti::config::LogFacility** (p. 459).
rti::core::policy::Monitoring (p. 1706)	**enable** (p. 1708)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.
rti::core::policy::Monitoring (p. 1706)	**application_name** (p. 1709)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The name of the resource that represents this RTI Connext application.
rti::core::policy::Monitoring (p. 1706)	**distribution_settings** (p. 1710)
	<<***extension***>> *(p. 116)* Configures the distribution of telemetry data.
rti::core::policy::Monitoring (p. 1706)	**telemetry_data** (p. 1711)
	<<***extension***>> *(p. 116)* Configures the telemetry data that will be distributed.
rti::core::LocatorFilterElement (p. 1678)	**locators** (p. 1679)
	<<***extension***>> *(p. 116)* Sequence containing from one to 16 Locator_t, used to specify the multicast address locators of an individual channel within a Multi↩ Channel DataWriter.
rti::core::LocatorFilterElement (p. 1678)	**filter_expression** (p. 1680)
	<<***extension***>> *(p. 116)* A logical expression used to determine the data that will be published in the channel.
rti::core::policy::LocatorFilter (p. 1676)	**locator_filters** (p. 1677)

	<<***extension***>> *(p. 116)* A sequence of **rti::core::LocatorFilterElement** (p. 1678). Each **rti::core::LocatorFilterElement** (p. 1678) reports the configuration of a single channel of a MultiChannel DataWriter.
rti::core::policy::LocatorFilter (p. 1676)	**filter_name** (p. 1678)
	<<***extension***>> *(p. 116)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.
rti::core::ChannelSettings (p. 921)	**multicast_settings** (p. 922)
	<<***extension***>> *(p. 116)* A sequence of **rti::core::TransportMulticastSettings** (p. 2549) used to configure the multicast addresses associated with a channel.
rti::core::ChannelSettings (p. 921)	**filter_expression** (p. 923)
	<<***extension***>> *(p. 116)* A logical expression used to determine the data that will be published in the channel.
rti::core::ChannelSettings (p. 921)	**priority** (p. 924)
	<<***extension***>> *(p. 116)* Publication priority
rti::core::policy::MultiChannel (p. 1749)	**channels** (p. 1751)
	<<***extension***>> *(p. 116)* A sequence of **rti::core::ChannelSettings** (p. 921) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.
rti::core::policy::MultiChannel (p. 1749)	**filter_name** (p. 1752)
	<<***extension***>> *(p. 116)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.
rti::core::policy::TopicQueryDispatch (p. 2522)	**enable** (p. 2524)
	<<***extension***>> *(p. 116)* Allows this writer to dispatch TopicQueries
rti::core::policy::TopicQueryDispatch (p. 2522)	**publication_period** (p. 2525)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Sets the periodic interval at which samples are published.
rti::core::policy::TopicQueryDispatch (p. 2522)	**samples_per_period** (p. 2525)
	<<***extension***>> *(p. 116)* Sets the maximum number of samples to publish in each publication_period.

6.47 Functional Categories

These categories help you to discover which QoS may be relevant or related to a desired functionality or feature. For example, if you are interested in **Sending Large Data** (p. 197), then looking at any QoS in that category will be helpful.

Modules

- **Application Acknowledgement**

  This page lists all QoS related to application acknowledgement.
- **Application-Defined Metadata**

  This page lists QoS fields that can be used to send application-specific data during participant and endpoint discovery.
- **Application Startup**

  Use these QoS fields to configure entity creation and application startup settings.
- **Batching**

  Use these QoS fields to configure the batching feature.
- **Concurrency**

  Use these QoS fields to configure threads and concurrency settings in RTI Connext.
- **Content Filtering**

  Use these QoS fields if you are using content filtering on your data.
- **Data Availability**

  Use these QoS fields to configure how much data, which data, and when that data is available to an application.
- **Debuggability**

  Use these QoS to configure logging and observability capabilities to help debug RTI Connext applications.
- **Discovery**

  Use these QoS fields to configure how applications discover each other.
- **Discovery Isolation**

  Use these QoS fields to isolate applications from each other.
- **Durability**

  Use these QoS fields to configure how much and how long data remains retrievable in the system.
- **Flow Control**

  Use these QoS fields to configure features which help to control the rate at which data is sent, preventing network congestion.
- **Fragmentation**

  QoS related to DDS-level data fragmentation.
- **Instance Lifecycle**

  QoS that control the lifecycle of instances.
- **Sending Large Data**

  Use these QoS fields to optimize the process of sending large data.
- **Liveliness**

  Use these QoS fields to manage the liveliness of participants and endpoints, ensuring active communication and timely updates of the system state.
- **Logging**

  Use these QoS fields to configure the RTI Connext logging infrastructure.
- **Middleware Memory Management**

  Use these QoS fields to configure the memory usage of the middleware, including various initial and max limits.
- **QoS Profile Management**

  Use these QoS fields to configure how QoS profiles are loaded and which ones should be used when entities are created.
- **Reliability QoS**

  Reliability related QoS.
- **Sample Memory Management QoS**

  Use these QoS fields to configure how Connext manages memory related to the samples that are stored and managed by the DataReader and DataWriter queues.
- **Sample Ordering**

Use these QoS fields to configure how samples are ordered when stored and accessed by a DataReader.

- **Security**

  Use these QoS fields to enable and configure some security-related capabilities in RTI Connext.
- **Throughput**

  Use these QoS fields to configure features which may be helpful to increase throughput.
- **Transports**

  Use these QoS fields to configure aspects of the transports (udp, shared memory, etc.) used for communication.
- **Topic Queries**

  Use these QoS fields to configure the topic query feature.
- **Type System**

  Use these QoS fields to configure aspects of the type system.

6.47.1 Detailed Description

These categories help you to discover which QoS may be relevant or related to a desired functionality or feature. For example, if you are interested in **Sending Large Data** (p. 197), then looking at any QoS in that category will be helpful.

6.48 Application Acknowledgement

This page lists all QoS related to application acknowledgement.

This page lists all QoS related to application acknowledgement.

See `Application Acknowledgment`, in the *Core Libraries User's Manual*, for more information about this feature.

Advanced Settings

dds::core::policy::Reliability (p. 2149)	**acknowledgment_kind** (p. 2154)
	<<***extension***>> *(p. 116)* Kind of reliable acknowledgment
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_app_ack_response_length** (p. 1101)
	<<***extension***>> *(p. 116)* Maximum length of application-level acknowledgment response data
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_app_ack_remote_readers** (p. 1245)
	<<***extension***>> *(p. 116)* The maximum number of application-level acknowledging remote readers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterProtocol (p. 1210)	**propagate_app_ack_with_no_response** (p. 1216)

	<<***extension***>> *(p. 116)* Controls whether or not a **dds::pub::DataWriter** (p. 1138) receives **dds::pub::DataWriterListener::on_application_acknowledgment** (p. 1207) notifications with an empty or invalid response.
rti::core::RtpsReliableReaderProtocol (p. 2221)	**app_ack_period** (p. 2227)
	<<***extension***>> *(p. 116)* The period at which application-level acknowledgment messages are sent
rti::core::RtpsReliableReaderProtocol (p. 2221)	**min_app_ack_response_keep_duration** (p. 2227)
	<<***extension***>> *(p. 116)* Minimum duration for which application-level acknowledgment response data is kept
rti::core::RtpsReliableReaderProtocol (p. 2221)	**samples_per_app_ack** (p. 2228)
	<<***extension***>> *(p. 116)* The minimum number of samples acknowledged by one application-level acknowledgment message

6.49 Application-Defined Metadata

This page lists QoS fields that can be used to send application-specific data during participant and endpoint discovery.

This page lists QoS fields that can be used to send application-specific data during participant and endpoint discovery.

There are also various resource limits related to these QoS which are also listed below.

Basic Settings

rti::core::policy::Property (p. 1969)	**value** (p. 1973)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Sequence of properties.

Advanced Settings

dds::core::policy::UserData (p. 2591)	**value** (p. 2593)
	<<***spec***>> *(p. 112)* a sequence of octets
dds::core::policy::GroupData (p. 1591)	**value** (p. 1593)
	<<***spec***>> *(p. 112)* a sequence of octets
rti::core::policy::TypeSupport (p. 2574)	**plugin_data** (p. 2575)
	[DEPRECATED] **(p. 296)** <<***extension***>> *(p. 116)* Value to pass into the type plugin's de-/serialization function

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**participant_user_data_max_length** (p. 1414)
	<<***extension***>> *(p. 116)* Maximum length of user data in **dds::domain::qos::DomainParticipantQos** (p. 1384) and **dds::topic::ParticipantBuiltinTopicData** (p. 1908)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**topic_data_max_length** (p. 1414)
	<<***extension***>> *(p. 116)* Maximum length of topic data in **dds::topic::qos::TopicQos** (p. 2509), **dds::topic::TopicBuiltinTopicData** (p. 2493), **dds::topic::PublicationBuiltinTopicData** (p. 1977) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**publisher_group_data_max_length** (p. 1415)
	<<***extension***>> *(p. 116)* Maximum length of group data in **dds::pub::qos::PublisherQos** (p. 2007) and **dds::topic::PublicationBuiltinTopicData** (p. 1977)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**subscriber_group_data_max_length** (p. 1415)
	<<***extension***>> *(p. 116)* Maximum length of group data in **dds::sub::qos::SubscriberQos** (p. 2423) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_user_data_max_length** (p. 1416)
	<<***extension***>> *(p. 116)* Maximum length of user data in **dds::pub::qos::DataWriterQos** (p. 1226) and **dds::topic::PublicationBuiltinTopicData** (p. 1977)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_user_data_max_length** (p. 1416)
	<<***extension***>> *(p. 116)* Maximum length of user data in **dds::sub::qos::DataReaderQos** (p. 1072) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**participant_property_list_max_length** (p. 1424)
	<<***extension***>> *(p. 116)* Maximum number of properties associated with the **dds::domain::DomainParticipant** (p. 1326).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**participant_property_string_max_length** (p. 1425)
	<<***extension***>> *(p. 116)* Maximum string length of the properties associated with the **dds::domain::DomainParticipant** (p. 1326).

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_property_list_max_length** (p. 1425)
	<<***extension***>> *(p. 116)* Maximum number of properties associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_property_string_max_length** (p. 1426)
	<<***extension***>> *(p. 116)* Maximum string length of the properties associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_property_list_max_length** (p. 1426)
	<<***extension***>> *(p. 116)* Maximum number of properties associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_property_string_max_length** (p. 1427)
	<<***extension***>> *(p. 116)* Maximum string length of the properties associated with a **dds::sub::DataReader** (p. 976).

6.50 Application Startup

Use these QoS fields to configure entity creation and application startup settings.

Use these QoS fields to configure entity creation and application startup settings.

Basic Settings

dds::core::policy::EntityFactory (p. 1527)	**autoenable_created_entities** (p. 1529)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Specifies whether the entity acting as a factory automatically enables the instances it creates.

6.51 Batching

Use these QoS fields to configure the batching feature.

Use these QoS fields to configure the batching feature.

See `BATCH QosPolicy (DDS Extension)`, in the *Core Libraries User's Manual*, for more information about this feature.

Basic Settings

rti::core::policy::Batch (p. 880)	**enable** (p. 882)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Specifies whether or not batching is enabled.
rti::core::policy::Batch (p. 880)	**max_data_bytes** (p. 882)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The maximum cumulative length of all serialized samples in a batch.
rti::core::policy::Batch (p. 880)	**max_samples** (p. 883)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The maximum number of samples in a batch.

Advanced Settings

rti::core::policy::AsynchronousPublisher (p. 832)	**disable_asynchronous_batch** (p. 836)
	[DEPRECATED] (p. 296) *<<**extension**>> (p. 116)* Disable asynchronous batch flushing.
rti::core::policy::AsynchronousPublisher (p. 832)	**asynchronous_batch_thread** (p. 837)
	*<<**extension**>> (p. 116)* Settings of the batch flushing thread.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_batches** (p. 1239)
	*<<**extension**>> (p. 116)* Represents the initial number of batches a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_batches** (p. 1240)
	*<<**extension**>> (p. 116)* Represents the maximum number of batches a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::Batch (p. 880)	**max_flush_delay** (p. 884)
	*<<**extension**>> (p. 116)* The maximum flush delay.
rti::core::policy::Batch (p. 880)	**source_timestamp_resolution** (p. 885)
	*<<**extension**>> (p. 116)* Batch source timestamp resolution.
rti::core::policy::Batch (p. 880)	**thread_safe_write** (p. 886)
	*<<**extension**>> (p. 116)* Determines whether or not the write operation is thread safe.

6.52 Concurrency

Use these QoS fields to configure threads and concurrency settings in RTI Connext.

Use these QoS fields to configure threads and concurrency settings in RTI Connext.

See `Connext Threading Model`, in the *Core Libraries User's Manual*, for more information.

This community article gives a thorough overview of how and why to use many of the QoS listed below: `https://community.rti.com/kb/achieving-low-jitter-performance-connext-pro`.

Advanced Settings

dds::core::policy::TransportPriority (p. 2552)	**value** (p. 2553)
	<<***spec***>> *(p. 112)* This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.
rti::core::policy::AsynchronousPublisher (p. 832)	**disable_asynchronous_write** (p. 835)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Disable asynchronous publishing.
rti::core::policy::AsynchronousPublisher (p. 832)	**thread** (p. 835)
	<<***extension***>> *(p. 116)* Settings of the publishing thread.
rti::core::policy::AsynchronousPublisher (p. 832)	**disable_asynchronous_batch** (p. 836)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Disable asynchronous batch flushing.
rti::core::policy::AsynchronousPublisher (p. 832)	**asynchronous_batch_thread** (p. 837)
	<<***extension***>> *(p. 116)* Settings of the batch flushing thread.
rti::core::policy::AsynchronousPublisher (p. 832)	**disable_topic_query_publication** (p. 838)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Disable topic query publication.
rti::core::policy::AsynchronousPublisher (p. 832)	**topic_query_publication_thread** (p. 838)
	<<***extension***>> *(p. 116)* Settings of the **rti::sub::TopicQuery** (p. 2516) publication thread.
rti::core::policy::Discovery (p. 1265)	**metatraffic_transport_priority** (p. 1269)
	<<***extension***>> *(p. 116)* The transport priority to use for the Discovery meta-traffic.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_concurrent_blocking_threads** (p. 1238)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The initial number of threads that are allowed to concurrently block on write call on the same **dds::pub::DataWriter** (p. 1138).

rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_concurrent_blocking_threads** (p. 1238)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum number of threads that are allowed to concurrently block on write call on the same **dds::pub::↩ DataWriter** (p. 1138).
rti::core::policy::Event (p. 1541)	**thread** (p. 1542)
	<<***extension***>> *(p. 116)* Event thread QoS.
rti::core::policy::Database (p. 969)	**thread** (p. 971)
	<<***extension***>> *(p. 116)* Database thread settings.
rti::core::policy::ReceiverPool (p. 2144)	**thread** (p. 2146)
	<<***extension***>> *(p. 116)* Receiver pool thread(s).
rti::core::policy::Batch (p. 880)	**thread_safe_write** (p. 886)
	<<***extension***>> *(p. 116)* Determines whether or not the write operation is thread safe.
rti::core::MonitoringEventDistributionSettings (p. 1722)	**concurrency_level** (p. 1723)
	<<***extension***>> *(p. 116)* Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
rti::core::MonitoringEventDistributionSettings (p. 1722)	**thread** (p. 1726)
	<<***extension***>> *(p. 116)* The settings of the event metric thread.
rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**thread** (p. 1743)
	<<***extension***>> *(p. 116)* The settings of the periodic metric thread.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**concurrency_level** (p. 1729)
	<<***extension***>> *(p. 116)* Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**thread** (p. 1731)
	<<***extension***>> *(p. 116)* The settings of the logging thread.

6.53 Content Filtering

Use these QoS fields if you are using content filtering on your data.

Use these QoS fields if you are using content filtering on your data.

Content filtering most commonly refers to using a **dds::topic::ContentFilteredTopic** (p. 954), but data can also be filtered using **rti::sub::TopicQuery** (p. 2516) to run a new filter over the data in a DataWriter's cache, or **dds::sub↩ ::cond::QueryCondition** (p. 2059) to read a subset of the data that is currently stored in the DataReader's cache.

See also

Queries and Filters Syntax (p. 84).

Advanced Settings

rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_query_condition_filters** (p. 1100)
	<<***extension***>> *(p. 116)* The maximum number of query condition filters a reader is allowed.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_remote_reader_filters** (p. 1239)
	<<***extension***>> *(p. 116)* The maximum number of remote DataReaders for which the **dds::pub::DataWriter** (p. 1138) will perform content-based filtering.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filtered_topic_allocation** (p. 1403)
	<<***extension***>> *(p. 116)* Allocation settings applied to content filtered topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filter_allocation** (p. 1404)
	<<***extension***>> *(p. 116)* Allocation settings applied to content filter.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**query_condition_allocation** (p. 1405)
	<<***extension***>> *(p. 116)* Allocation settings applied to query condition pool.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filtered_topic_hash_buckets** (p. 1412)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Number of hash buckets for content filtered topics.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filter_hash_buckets** (p. 1412)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Number of hash buckets for content filters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**contentfilter_property_max_length** (p. 1422)

	<<***extension***>> *(*p. *116)* This field is the maximum length of all data related to a Content-filtered topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**channel_seq_max_length** (p. 1423)
	<<***extension***>> *(*p. *116)* Maximum number of channels that can be specified in **rti::core::policy::Multi↩ Channel** (p. 1749) for MultiChannel DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**channel_filter_expression_max_length** (p. 1424)
	<<***extension***>> *(*p. *116)* Maximum length of a channel **rti::core::ChannelSettings::filter_expression** (p. 923) in a MultiChannel DataWriter.
rti::core::ChannelSettings (p. 921)	**multicast_settings** (p. 922)
	<<***extension***>> *(*p. *116)* A sequence of **rti::core::↩ TransportMulticastSettings** (p. 2549) used to configure the multicast addresses associated with a channel.
rti::core::ChannelSettings (p. 921)	**filter_expression** (p. 923)
	<<***extension***>> *(*p. *116)* A logical expression used to determine the data that will be published in the channel.
rti::core::policy::MultiChannel (p. 1749)	**filter_name** (p. 1752)
	<<***extension***>> *(*p. *116)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

6.54 Data Availability

Use these QoS fields to configure how much data, which data, and when that data is available to an application.

Use these QoS fields to configure how much data, which data, and when that data is available to an application.

Data availability refers to configurations such as ownership and lifespan.

Basic Settings

dds::core::policy::Durability (p. 1433)	**kind** (p. 1437)
	<<***basic***>> *(*p. *104)* <<***spec***>> *(*p. *112)* The kind of durability.
dds::core::policy::Durability (p. 1433)	**writer_depth** (p. 1439)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Indicates the number of samples per instance that a durable **dds::pub::DataWriter** (p. 1138) will send to a late-joining **dds::sub::DataReader** (p. 976).

dds::core::policy::Deadline (p. 1255)	**period** (p. 1258)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Duration of the deadline period.
dds::core::policy::Ownership (p. 1899)	**kind** (p. 1904)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of ownership
dds::core::policy::OwnershipStrength (p. 1906)	**value** (p. 1908)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The strength value used to arbitrate among multiple writers.
dds::core::policy::TimeBasedFilter (p. 2469)	**minimum_separation** (p. 2472)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The minimum separation duration between subsequent samples.
dds::core::policy::History (p. 1602)	**kind** (p. 1604)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Specifies the kind of history to be kept.
dds::core::policy::History (p. 1602)	**depth** (p. 1605)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Specifies the number of samples per instance to be kept, when the `kind` is **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607)
dds::core::policy::ResourceLimits (p. 2203)	**max_samples** (p. 2205)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Represents the maximum samples the middleware can store for any one **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)).
dds::core::policy::ResourceLimits (p. 2203)	**max_instances** (p. 2206)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Represents the maximum number of instances a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::Lifespan (p. 1635)	**duration** (p. 1637)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Maximum duration for the data's validity.
rti::core::MonitoringEventDistributionSettings (p. 1722)	**publication_period** (p. 1726)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.

rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**polling_period** (p. 1744)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Period at which the periodic metric thread polls and publishes the periodic metrics.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**publication_period** (p. 1732)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Period at which the logging thread publishes log messages.

Advanced Settings

rti::core::policy::Availability (p. 868)	**enable_required_subscriptions** (p. 871)
	*<<**extension**>> (p. 116)* Enables support for required subscriptions in a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::Availability (p. 868)	**max_data_availability_waiting_time** (p. 871)
	*<<**extension**>> (p. 116)* Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
rti::core::policy::Availability (p. 868)	**max_endpoint_availability_waiting_time** (p. 872)
	*<<**extension**>> (p. 116)* Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
rti::core::policy::Availability (p. 868)	**required_matched_endpoint_groups** (p. 873)
	*<<**extension**>> (p. 116)* A sequence of endpoint groups.
dds::core::policy::Durability (p. 1433)	**direct_communication** (p. 1438)
	*<<**extension**>> (p. 116)* Indicates whether or not a TRANSIENT or PERSISTENT **dds::sub::DataReader** (p. 976) should receive samples directly from a TRANSIENT or PERSISTENT **dds::pub::DataWriter** (p. 1138)
dds::core::policy::Durability (p. 1433)	**storage_settings** (p. 1440)
	*<<**extension**>> (p. 116)* Configures durable writer history and durable reader state.
dds::core::policy::DurabilityService (p. 1443)	**service_cleanup_delay** (p. 1445)
	*<<**spec**>> (p. 112)* Controls when the service is able to remove all information regarding a data instances.
dds::core::policy::DurabilityService (p. 1443)	**max_samples** (p. 1446)
	*<<**spec**>> (p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.

dds::core::policy::DurabilityService (p. 1443)	**max_instances** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_samples_per_instance** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::Presentation (p. 1942)	**access_scope** (p. 1947)
	<<***spec***>> *(p. 112)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
dds::core::policy::Presentation (p. 1942)	**coherent_access** (p. 1947)
	<<***spec***>> *(p. 112)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.
dds::core::policy::Presentation (p. 1942)	**ordered_access** (p. 1948)
	<<***spec***>> *(p. 112)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
dds::core::policy::Presentation (p. 1942)	**drop_incomplete_coherent_set** (p. 1949)
	<<***extension***>> *(p. 116)* Indicates whether or not a **dds::sub::DataReader** (p. 976) should drop samples from an incomplete coherent set (one for which not all the samples were received). Such samples are reported as lost in the SAMPLE_LOST Status.
dds::core::policy::LatencyBudget (p. 1631)	**duration** (p. 1633)
	<<***spec***>> *(p. 112)* Duration of the maximum acceptable delay.
dds::core::policy::ResourceLimits (p. 2203)	**max_samples_per_instance** (p. 2207)
	<<***spec***>> *(p. 112)* Represents the maximum number of samples of any one instance a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_nowriter_samples_delay** (p. 2140)
	<<***spec***>> *(p. 112)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain samples for an instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_↩ no_writers()** (p. 1617).
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_disposed_samples_delay** (p. 2141)

	<<***spec***>> *(p. 112)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain samples for an instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617).
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_disposed_instances_delay** (p. 2142)
	<<***extension***>> *(p. 116)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain information about a received instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) and there are no samples for the instance in the DataReader queue.
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_nowriter_instances_delay** (p. 2143)
	<<***extension***>> *(p. 116)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain information about a received instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) and there are no samples for the instance in the DataReader queue.
rti::core::policy::TransportMulticastMapping (p. 2547)	**value** (p. 2549)
	<<***extension***>> *(p. 116)* A sequence of multicast communication mappings.
rti::core::policy::TransportMulticast (p. 2543)	**kind** (p. 2546)
	<<***extension***>> *(p. 116)* A value that specifies a way to determine how to obtain the multicast address.
rti::core::policy::DataWriterProtocol (p. 1210)	**virtual_guid** (p. 1212)
	<<***extension***>> *(p. 116)* The virtual GUID (Global Unique Identifier).
rti::core::policy::DataWriterProtocol (p. 1210)	**push_on_write** (p. 1213)
	<<***extension***>> *(p. 116)* Whether to push sample out when write is called.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_outstanding_reads** (p. 911)
	<<***extension***>> *(p. 116)* The initial number of outstanding reads that have not called finish yet on the same built-in topic **dds::sub::DataReader** (p. 976).
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_outstanding_reads** (p. 912)
	<<***extension***>> *(p. 116)* The maximum number of outstanding reads that have not called finish yet on the same built-in topic **dds::sub::DataReader** (p. 976).

rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_samples_per_read** (p. 912)
	*<<**extension**>> (p. 116)* Maximum number of samples that can be read/taken on a same built-in topic **dds::sub::DataReader** (p. 976).
rti::core::policy::Batch (p. 880)	**source_timestamp_resolution** (p. 885)
	*<<**extension**>> (p. 116)* Batch source timestamp resolution.
rti::core::ChannelSettings (p. 921)	**priority** (p. 924)
	*<<**extension**>> (p. 116)* Publication priority
rti::core::policy::MultiChannel (p. 1749)	**channels** (p. 1751)
	*<<**extension**>> (p. 116)* A sequence of **rti::core::ChannelSettings** (p. 921) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.
rti::core::policy::MultiChannel (p. 1749)	**filter_name** (p. 1752)
	*<<**extension**>> (p. 116)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

6.55 Debuggability

Use these QoS to configure logging and observability capabilities to help debug RTI Connext applications.

Use these QoS to configure logging and observability capabilities to help debug RTI Connext applications.

Basic Settings

rti::core::policy::EntityName (p. 1530)	**name** (p. 1531)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The name of the entity.
rti::config::Logger (p. 1683)	**verbosity** (p. 1685)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The verbosities at which RTI Connext diagnostic information is logged.
rti::config::Logger (p. 1683)	**category** (p. 1685)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Categories of logged messages.
rti::config::Logger (p. 1683)	**output_file** (p. 1686)

	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Specifies the file to which log messages will be redirected to.
rti::config::Logger (p. 1683)	**output_file_suffix** (p. 1687)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sets the file suffix when logging to a set of files
rti::core::MonitoringEventDistributionSettings (p. 1722)	**publication_period** (p. 1726)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**polling_period** (p. 1744)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Period at which the periodic metric thread polls and publishes the periodic metrics.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**max_historical_logs** (p. 1729)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**publication_period** (p. 1732)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Period at which the logging thread publishes log messages.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**enable** (p. 1713)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Enables the use of a dedicated **dds::domain::Domain↩ Participant** (p. 1326) to distribute the RTI Connext application telemetry data.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**domain_id** (p. 1713)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The domain ID used in the creation of RTI Monitoring Library 2.0 **dds::domain::DomainParticipant** (p. 1326).
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**collector_initial_peers** (p. 1715)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **rti::core::MonitoringDistribution↩ Settings::dedicated_participant** (p. 1719).

rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**middleware_forwarding_level** (p. 1734)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) Log messages with **rti::config::LogFacility::middleware** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**security_event_forwarding_level** (p. 1734)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) Log messages with **rti::config::LogFacility::security_event** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**service_forwarding_level** (p. 1735)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) Log messages with **rti::config::LogFacility::service** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**user_forwarding_level** (p. 1736)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) Log messages with **rti::config::LogFacility::user** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringTelemetryData (p. 1744)	**metrics** (p. 1745)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) Sequence of **rti::core::MonitoringMetricSelection** (p. 1736) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.
rti::core::MonitoringTelemetryData (p. 1744)	**logs** (p. 1746)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) **rti::core::MonitoringLoggingForwardingSettings** (p. 1733) containing the **rti::config::SyslogVerbosity** (p. 458) levels that will be forwarded for the different **rti::config::LogFacility** (p. 459).
rti::core::policy::Monitoring (p. 1706)	**enable** (p. 1708)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.
rti::core::policy::Monitoring (p. 1706)	**application_name** (p. 1709)

	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The name of the resource that represents this RTI Connext application.

Advanced Settings

rti::core::policy::DiscoveryConfig (p. 1271)	**default_domain_announcement_period** (p. 1300)
	<<***extension***>> *(p. 116)* The period to announce a participant to the default domain 0.
rti::core::policy::DiscoveryConfig (p. 1271)	**ignore_default_domain_announcements** (p. 1301)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.
rti::core::policy::EntityName (p. 1530)	**role_name** (p. 1532)
	<<***extension***>> *(p. 116)* The entity role name.
rti::core::policy::Service (p. 2350)	**kind** (p. 2354)
	<<***extension***>> *(p. 116)* The kind of service.
rti::config::Logger (p. 1683)	**print_format** (p. 1688)
	<<***extension***>> *(p. 116)* The format used to output RTI Connext diagnostic information.
rti::config::Logger (p. 1683)	**max_bytes_per_file** (p. 1687)
	<<***extension***>> *(p. 116)* Specifies the maximum number of bytes a single file can contain
rti::config::Logger (p. 1683)	**max_files** (p. 1687)
	<<***extension***>> *(p. 116)* Specifies the maximum number of files to create before overwritting the previous ones.
rti::core::MonitoringEventDistributionSettings (p. 1722)	**concurrency_level** (p. 1723)
	<<***extension***>> *(p. 116)* Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
rti::core::MonitoringEventDistributionSettings (p. 1722)	**datawriter_qos_profile_name** (p. 1724)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes event metrics.
rti::core::MonitoringEventDistributionSettings (p. 1722)	**thread** (p. 1726)

	<<***extension***>> *(p. 116)* The settings of the event metric thread.
rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**datawriter_qos_profile_name** (p. 1742)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes periodic metrics.
rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**thread** (p. 1743)
	<<***extension***>> *(p. 116)* The settings of the periodic metric thread.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**concurrency_level** (p. 1729)
	<<***extension***>> *(p. 116)* Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**datawriter_qos_profile_name** (p. 1730)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes log messages.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**thread** (p. 1731)
	<<***extension***>> *(p. 116)* The settings of the logging thread.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**participant_qos_profile_name** (p. 1714)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the **dds::domain::DomainParticipant** (p. 1326) that will be used to distribute telemetry data.
rti::core::MonitoringDistributionSettings (p. 1716)	**dedicated_participant** (p. 1719)
	<<***extension***>> *(p. 116)* Configures the use of a dedicated **dds::domain::DomainParticipant** (p. 1326) to distribute the RTI Connext application telemetry data.
rti::core::MonitoringDistributionSettings (p. 1716)	**publisher_qos_profile_name** (p. 1718)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.
rti::core::MonitoringDistributionSettings (p. 1716)	**event_settings** (p. 1720)
	<<***extension***>> *(p. 116)* Configures the distribution of event metrics.

rti::core::MonitoringDistributionSettings (p. 1716)	**periodic_settings** (p. 1720)
	<<***extension***>> *(p. 116)* Configures the distribution of periodic metrics.
rti::core::MonitoringDistributionSettings (p. 1716)	**logging_settings** (p. 1721)
	<<***extension***>> *(p. 116)* Configures the distribution of logging messages.
rti::core::MonitoringMetricSelection (p. 1736)	**resource_selection** (p. 1737)
	<<***extension***>> *(p. 116)* An expression pattern that selects a subset of resources by matching the pattern to the resource names. **rti::core::MonitoringMetric↩ Selection::enabled_metrics_selection** (p. 1738) and **rti::core::MonitoringMetricSelection::disabled_↩ metrics_selection** (p. 1739) are applied to this subset of resources.
rti::core::MonitoringMetricSelection (p. 1736)	**enabled_metrics_selection** (p. 1738)
	<<***extension***>> *(p. 116)* A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by **rti::core::Monitoring↩ MetricSelection::resource_selection** (p. 1737).
rti::core::MonitoringMetricSelection (p. 1736)	**disabled_metrics_selection** (p. 1739)
	<<***extension***>> *(p. 116)* A sequence of POSIX fnmatch patterns that mach the names of the metrics that should not be collected and distributed for the observable resources selected by **rti::core::Monitoring↩ MetricSelection::resource_selection** (p. 1737).
rti::core::policy::Monitoring (p. 1706)	**distribution_settings** (p. 1710)
	<<***extension***>> *(p. 116)* Configures the distribution of telemetry data.
rti::core::policy::Monitoring (p. 1706)	**telemetry_data** (p. 1711)
	<<***extension***>> *(p. 116)* Configures the telemetry data that will be distributed.

6.56 Discovery

Use these QoS fields to configure how applications discover each other.

Use these QoS fields to configure how applications discover each other.

See `Discovery Overview`, in the *Core Libraries User's Manual*, for more information about discovery.

Basic Settings

rti::core::policy::WireProtocol (p. 2630)	**participant_id** (p. 2635)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* A value used to distinguish among different participants belonging to the same domain on the same host.
rti::core::policy::Discovery (p. 1265)	**initial_peers** (p. 1268)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.
rti::core::policy::Discovery (p. 1265)	**multicast_receive_addresses** (p. 1268)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the Domain↩ Participant.
rti::core::policy::Discovery (p. 1265)	**accept_unknown_peers** (p. 1270)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Whether to accept a new participant that is not in the initial peers list.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_liveliness_lease_duration** (p. 1279)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The liveliness lease duration for the participant.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_liveliness_assert_period** (p. 1279)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The period to assert liveliness for the participant.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_announcement_period** (p. 1280)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
rti::core::policy::DiscoveryConfig (p. 1271)	**initial_participant_announcements** (p. 1282)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The number of initial announcements sent when a participant is first enabled.
rti::core::policy::DiscoveryConfig (p. 1271)	**new_remote_participant_announcements** (p. 1283)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The number of participant announcements sent when a remote participant is newly discovered.
rti::core::policy::DiscoveryConfig (p. 1271)	**min_initial_participant_announcement_period** (p. 1283)

	<<***basic***>> (p. *104)* <<***extension***>> (p. *116)* The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
rti::core::policy::DiscoveryConfig (p. 1271)	**max_initial_participant_announcement_period** (p. 1284)
	<<***basic***>> (p. *104)* <<***extension***>> (p. *116)* The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**domain_id** (p. 1713)
	<<***basic***>> (p. *104)* <<***extension***>> (p. *116)* The domain ID used in the creation of RTI Monitoring Library 2.0 **dds::domain::DomainParticipant** (p. 1326).
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**collector_initial_peers** (p. 1715)
	<<***basic***>> (p. *104)* <<***extension***>> (p. *116)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **rti::core::MonitoringDistribution↩ Settings::dedicated_participant** (p. 1719).

Advanced Settings

rti::core::policy::WireProtocol (p. 2630)	**rtps_host_id** (p. 2636)
	<<***extension***>> (p. *116)* The RTPS Host ID of the domain participant.
rti::core::policy::WireProtocol (p. 2630)	**rtps_app_id** (p. 2636)
	<<***extension***>> (p. *116)* The RTPS App ID of the domain participant.
rti::core::policy::WireProtocol (p. 2630)	**rtps_instance_id** (p. 2637)
	<<***extension***>> (p. *116)* The RTPS Instance ID of the **dds↩ ::domain::DomainParticipant** (p. 1326).
rti::core::policy::WireProtocol (p. 2630)	**rtps_well_known_ports** (p. 2638)
	<<***extension***>> (p. *116)* Configures the RTPS well-known port mappings.
rti::core::policy::WireProtocol (p. 2630)	**rtps_reserved_port_mask** (p. 2639)
	<<***extension***>> (p. *116)* Specifies which well-known ports to reserve when enabling the participant.

rti::core::policy::WireProtocol (p. 2630)	**rtps_auto_id_kind** (p. 2639)
	<<***extension***>> *(p. 116)* Kind of auto mechanism used to calculate the GUID prefix.
rti::core::policy::Discovery (p. 1265)	**enabled_transports** (p. 1267)
	<<***extension***>> *(p. 116)* The transports available for use by the Discovery mechanism.
rti::core::policy::Discovery (p. 1265)	**metatraffic_transport_priority** (p. 1269)
	<<***extension***>> *(p. 116)* The transport priority to use for the Discovery meta-traffic.
rti::core::policy::Discovery (p. 1265)	**enable_endpoint_discovery** (p. 1271)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Whether to automatically enable endpoint discovery for all the remote participants.
rti::core::policy::DiscoveryConfig (p. 1271)	**remote_participant_purge_kind** (p. 1281)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
rti::core::policy::DiscoveryConfig (p. 1271)	**max_liveliness_loss_detection_period** (p. 1281)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_reader_resource_limits** (p. 1285)
	<<***extension***>> *(p. 116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_reader** (p. 1286)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_reader_resource_limits** (p. 1287)
	<<***extension***>> *(p. 116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_reader** (p. 1288)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_reader_resource_limits** (p. 1289)
	<<***extension***>> *(p. 116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_writer** (p. 1296)

	<<***extension***>> *(*p. *116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core::policy::ReliabilityKind_def** (p. 2156).
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_writer_publish_mode** (p. 1298)
	<<***extension***>> *(*p. *116)* Publish mode policy for the built-in publication writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_writer_publish_mode** (p. 1298)
	<<***extension***>> *(*p. *116)* Publish mode policy for the built-in subscription writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**asynchronous_publisher** (p. 1299)
	<<***extension***>> *(*p. *116)* Asynchronous publishing settings for the discovery **dds::pub::Publisher** (p. 1993) and all entities that are created by it.
rti::core::policy::DiscoveryConfig (p. 1271)	**default_domain_announcement_period** (p. 1300)
	<<***extension***>> *(*p. *116)* The period to announce a participant to the default domain 0.
rti::core::policy::DiscoveryConfig (p. 1271)	**ignore_default_domain_announcements** (p. 1301)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_reader_reliability_kind** (p. 1295)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Reliability policy for a built-in participant message reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_reader** (p. 1295)
	<<***extension***>> *(*p. *116)* RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **rti::core::policy::DiscoveryConfig::participant_message_reader_reliability_kind** (p. 1295) is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157).
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_writer** (p. 1289)
	<<***extension***>> *(*p. *116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_writer_data_lifecycle** (p. 1291)
	<<***extension***>> *(*p. *116)* Writer data lifecycle settings for a built-in publication writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_writer** (p. 1291)

	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_writer_data_lifecycle** (p. 1293)
	<<***extension***>> *(p. 116)* Writer data lifecycle settings for a built-in subscription writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**builtin_discovery_plugins** (p. 1293)
	<<***extension***>> *(p. 116)* Mask of built-in discovery plugin kinds.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_writer** (p. 1301)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **rti::topic::ServiceRequest** (p. 2358) writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_writer_data_lifecycle** (p. 1303)
	<<***extension***>> *(p. 116)* Writer data lifecycle settings for a built-in **rti::topic::ServiceRequest** (p. 2358) writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_writer_publish_mode** (p. 1303)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in service request writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_reader** (p. 1304)
	<<***extension***>> *(p. 116)* RTPS reliable reader protocol-related configuration settings for a built-in **rti::topic::ServiceRequest** (p. 2358) reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_writer** (p. 1307)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_writer_publish_mode** (p. 1309)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in secure volatile writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_reader** (p. 1309)
	<<***extension***>> *(p. 116)* RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**endpoint_type_object_lb_serialization_threshold** (p. 1310)
	<<***extension***>> *(p. 116)* Controls when/if the TypeObject v1 is compressed when propagated in Simple Endpoint Discovery.
rti::core::policy::DiscoveryConfig (p. 1271)	**dns_tracker_polling_period** (p. 1311)
	<<***extension***>> *(p. 116)* Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.

rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_reader** (p. 1312)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_reader_resource_limits** (p. 1313)
	<<***extension***>> *(p. 116)* Resource limits for the built-in topic participant configuration reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_writer_publish_mode** (p. 1316)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in participant configuration writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_writer** (p. 1314)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_writer_data_lifecycle** (p. 1316)
	<<***extension***>> *(p. 116)* Writer data lifecycle settings for a built-in participant configuration writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**request_types_filter** (p. 1317)
	<<***extension***>> *(p. 116)* Topic name pattern for requesting all types via Type Lookup Service.

6.57 Discovery Isolation

Use these QoS fields to isolate applications from each other.

Use these QoS fields to isolate applications from each other.

Applications can be isolated through mechanisms such as domain ids, domain tags, and partitions.

See `Discovery Overview`, in the *Core Libraries User's Manual*, for more information about discovery.

Basic Settings

dds::core::policy::Partition (p. 1923)	**name** (p. 1926)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* A list of partition names
rti::core::policy::Discovery (p. 1265)	**initial_peers** (p. 1268)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.

rti::core::policy::Discovery (p. 1265)	**accept_unknown_peers** (p. 1270)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Whether to accept a new participant that is not in the initial peers list.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**domain_id** (p. 1713)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The domain ID used in the creation of RTI Monitoring Library 2.0 **dds::domain::DomainParticipant** (p. 1326).
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**collector_initial_peers** (p. 1715)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **rti::core::MonitoringDistribution↩ Settings::dedicated_participant** (p. 1719).

Advanced Settings

rti::core::policy::Discovery (p. 1265)	**enable_endpoint_discovery** (p. 1271)
	[DEPRECATED] (p. **296)** <<***extension***>> *(p. 116)* Whether to automatically enable endpoint discovery for all the remote participants.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_partitions** (p. 1417)
	<<***extension***>> *(p. 116)* Maximum number of partition name strings allowable in a **dds::core::policy::Partition** (p. 1923).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_partition_cumulative_characters** (p. 1417)
	<<***extension***>> *(p. 116)* Maximum number of combined characters allowable in all partition names in a **dds::core::policy::Partition** (p. 1923).

6.58 Durability

Use these QoS fields to configure how much and how long data remains retrievable in the system.

Use these QoS fields to configure how much and how long data remains retrievable in the system.

Basic Settings

dds::core::policy::Durability (p. 1433)	**kind** (p. 1437)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* The kind of durability.
dds::core::policy::Durability (p. 1433)	**writer_depth** (p. 1439)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Indicates the number of samples per instance that a durable **dds::pub::Data↩ Writer** (p. 1138) will send to a late-joining **dds::sub::DataReader** (p. 976).
dds::core::policy::ResourceLimits (p. 2203)	**max_samples** (p. 2205)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Represents the maximum samples the middleware can store for any one **dds::pub::↩ DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)).
dds::core::policy::ResourceLimits (p. 2203)	**max_instances** (p. 2206)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Represents the maximum number of instances a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::Lifespan (p. 1635)	**duration** (p. 1637)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Maximum duration for the data's validity.

Advanced Settings

rti::core::policy::Availability (p. 868)	**enable_required_subscriptions** (p. 871)
	<<***extension***>> *(p. 116)* Enables support for required subscriptions in a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::Availability (p. 868)	**max_data_availability_waiting_time** (p. 871)
	<<***extension***>> *(p. 116)* Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
rti::core::policy::Availability (p. 868)	**max_endpoint_availability_waiting_time** (p. 872)
	<<***extension***>> *(p. 116)* Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
rti::core::policy::Availability (p. 868)	**required_matched_endpoint_groups** (p. 873)
	<<***extension***>> *(p. 116)* A sequence of endpoint groups.
dds::core::policy::Durability (p. 1433)	**direct_communication** (p. 1438)

	<<***extension***>> *(*p. *116)* Indicates whether or not a TRANSIENT or PERSISTENT **dds::sub::DataReader** (p. 976) should receive samples directly from a TRANSIENT or PERSISTENT **dds::pub::DataWriter** (p. 1138)
dds::core::policy::Durability (p. 1433)	**storage_settings** (p. 1440)
	<<***extension***>> *(*p. *116)* Configures durable writer history and durable reader state.
rti::core::PersistentStorageSettings (p. 1927)	**enable** (p. 1929)
	<<***extension***>> *(*p. *116)* Enables durable writer history in a **dds::pub::DataWriter** (p. 1138) and durable reader state in a **dds::sub::DataReader** (p. 976).
rti::core::PersistentStorageSettings (p. 1927)	**file_name** (p. 1929)
	<<***extension***>> *(*p. *116)* The file name where the durable writer history or durable reader state will be stored.
rti::core::PersistentStorageSettings (p. 1927)	**trace_file_name** (p. 1931)
	<<***extension***>> *(*p. *116)* The file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.
rti::core::PersistentStorageSettings (p. 1927)	**journal_kind** (p. 1932)
	<<***extension***>> *(*p. *116)* Sets the journal mode of the persistent storage.
rti::core::PersistentStorageSettings (p. 1927)	**synchronization_kind** (p. 1933)
	<<***extension***>> *(*p. *116)* Sets the level of synchronization with the physical disk.
rti::core::PersistentStorageSettings (p. 1927)	**vacuum** (p. 1933)
	<<***extension***>> *(*p. *116)* Sets the auto-vacuum status of the storage.
rti::core::PersistentStorageSettings (p. 1927)	**restore** (p. 1934)
	<<***extension***>> *(*p. *116)* Indicates if the persisted writer history or reader state must be restored.
rti::core::PersistentStorageSettings (p. 1927)	**writer_instance_cache_allocation** (p. 1934)
	<<***extension***>> *(*p. *116)* Configures the resource limits associated with the instance durable writer history cache.
rti::core::PersistentStorageSettings (p. 1927)	**writer_sample_cache_allocation** (p. 1935)
	<<***extension***>> *(*p. *116)* Configures the resource limits associated with the sample durable writer history cache.
rti::core::PersistentStorageSettings (p. 1927)	**writer_memory_state** (p. 1936)

	<<***extension***>> *(p. 116)* Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.
rti::core::PersistentStorageSettings (p. 1927)	**reader_checkpoint_frequency** (p. 1937)
	<<***extension***>> *(p. 116)* Controls how often the reader state is stored into the database.
dds::core::policy::DurabilityService (p. 1443)	**service_cleanup_delay** (p. 1445)
	<<***spec***>> *(p. 112)* Controls when the service is able to remove all information regarding a data instances.
dds::core::policy::DurabilityService (p. 1443)	**history_kind** (p. 1445)
	<<***spec***>> *(p. 112)* The kind of history to apply in recouping durable data.
dds::core::policy::DurabilityService (p. 1443)	**history_depth** (p. 1446)
	<<***spec***>> *(p. 112)* Setting to use for the **dds::core::policy::Durability::writer_depth** (p. 1439) when recouping durable data.
dds::core::policy::DurabilityService (p. 1443)	**max_samples** (p. 1446)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_instances** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_samples_per_instance** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::ResourceLimits (p. 2203)	**max_samples_per_instance** (p. 2207)
	<<***spec***>> *(p. 112)* Represents the maximum number of samples of any one instance a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
rti::core::policy::DataWriterProtocol (p. 1210)	**propagate_app_ack_with_no_response** (p. 1216)
	<<***extension***>> *(p. 116)* Controls whether or not a **dds::pub::DataWriter** (p. 1138) receives **dds::pub::DataWriterListener::on_application_acknowledgment** (p. 1207) notifications with an empty or invalid response.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 2247)

	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Enables dynamic adjustment of sample keep duration in response to congestion.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_min_sample_keep_duration** (p. 2245)
	<<***extension***>> *(p. 116)* The minimum duration a sample is queued for ACK-disabled readers.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_max_sample_keep_duration** (p. 2246)
	<<***extension***>> *(p. 116)* The maximum duration a sample is queued for ACK-disabled readers.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_decrease_sample_keep_↩ duration_factor** (p. 2248)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Controls rate of contraction of dynamic sample keep duration.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_increase_sample_keep_↩ duration_factor** (p. 2248)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Controls rate of growth of dynamic sample keep duration.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_samples** (p. 909)
	<<***extension***>> *(p. 116)* Initial number of samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_samples** (p. 909)
	<<***extension***>> *(p. 116)* Maximum number of samples.
rti::core::policy::EntityName (p. 1530)	**role_name** (p. 1532)
	<<***extension***>> *(p. 116)* The entity role name.

6.59 Flow Control

Use these QoS fields to configure features which help to control the rate at which data is sent, preventing network congestion.

Use these QoS fields to configure features which help to control the rate at which data is sent, preventing network congestion.

Basic Settings

dds::core::policy::Reliability (p. 2149)	**max_blocking_time** (p. 2153)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The maximum time a DataWriter can block on a **write()** (p. 1179) call.
rti::core::policy::PublishMode (p. 2013)	**kind** (p. 2016)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Publishing mode
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**min_send_window_size** (p. 2249)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Minimum size of send window of unacknowledged samples
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_send_window_size** (p. 2250)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Maximum size of send window of unacknowledged samples

Advanced Settings

rti::core::policy::PublishMode (p. 2013)	**flow_controller_name** (p. 2016)
	*<<**extension**>> (p. 116)* Name of the associated flow controller
rti::core::policy::PublishMode (p. 2013)	**priority** (p. 2017)
	*<<**extension**>> (p. 116)* Publication priority
rti::core::policy::AsynchronousPublisher (p. 832)	**disable_asynchronous_write** (p. 835)
	[DEPRECATED] (p. 296) *<<**extension**>> (p. 116)* Disable asynchronous publishing.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**outstanding_asynchronous_sample_allocation** (p. 1405)
	*<<**extension**>> (p. 116)* Allocation settings applied to the maximum number of samples (from all **dds::pub::↩ DataWriter** (p. 1138)) waiting to be asynchronously written.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**flow_controller_allocation** (p. 1406)
	*<<**extension**>> (p. 116)* Allocation settings applied to flow controllers.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**flow_controller_hash_buckets** (p. 1413)

	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for flow controllers.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_writer_publish_mode** (p. 1298)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in publication writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_writer_publish_mode** (p. 1298)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in subscription writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**asynchronous_publisher** (p. 1299)
	<<***extension***>> *(p. 116)* Asynchronous publishing settings for the discovery **dds::pub::Publisher** (p. 1993) and all entities that are created by it.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_writer_publish_mode** (p. 1303)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in service request writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_writer_publish_mode** (p. 1309)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in secure volatile writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_writer_publish_mode** (p. 1316)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in participant configuration writer.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_bytes_per_nack_response** (p. 2244)
	<<***extension***>> *(p. 116)* The maximum total message size when resending rejected samples or data fragments.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**send_window_update_period** (p. 2251)
	<<***extension***>> *(p. 116)* Period in which send window may be dynamically changed
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**send_window_increase_factor** (p. 2252)
	<<***extension***>> *(p. 116)* Increases send window size by this percentage when reacting dynamically to network conditions
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**send_window_decrease_factor** (p. 2253)
	<<***extension***>> *(p. 116)* Decreases send window size by this percentage when reacting dynamically to network conditions

rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_repair_piggyback_heartbeat** (p. 2255)
	<<***extension***>> *(p. 116)* Prevents piggyback heartbeats from being sent with repair samples

6.60 Fragmentation

QoS related to DDS-level data fragmentation.

QoS related to DDS-level data fragmentation.

See `Data Fragmentation`, in the *Core Libraries User's Manual*, for more information about DDS-level fragmentation.

Advanced Settings

rti::core::policy::DataReaderResourceLimits (p. 1081)	**disable_fragmentation_support** (p. 1092)
	[DEPRECATED] (p. **296)** <<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) can receive fragmented samples.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragmented_samples** (p. 1092)
	<<***extension***>> *(p. 116)* The maximum number of samples for which the **dds::sub::DataReader** (p. 976) may store fragments at a given point in time.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_fragmented_samples** (p. 1093)
	<<***extension***>> *(p. 116)* The initial number of samples for which a **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragmented_samples_per_remote_writer** (p. 1094)
	<<***extension***>> *(p. 116)* The maximum number of samples per remote writer for which a **dds::sub::Data↩ Reader** (p. 976) may store fragments.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragments_per_sample** (p. 1094)
	<<***extension***>> *(p. 116)* Maximum number of fragments for a single sample.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**dynamically_allocate_fragmented_samples** (p. 1095)

	<<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**min_nack_response_delay** (p. 2242)
	<<***extension***>> *(p. 116)* The minimum delay to respond to a NACK or NACK_FRAG.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_nack_response_delay** (p. 2242)
	<<***extension***>> *(p. 116)* The maximum delay to respond to a NACK or NACK_FRAG.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**nack_suppression_duration** (p. 2243)
	<<***extension***>> *(p. 116)* The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_bytes_per_nack_response** (p. 2244)
	<<***extension***>> *(p. 116)* The maximum total message size when resending rejected samples or data fragments.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**disable_fragmentation_support** (p. 913)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Determines whether the built-in topic **dds::sub::Data↩ Reader** (p. 976) can receive fragmented samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragmented_samples** (p. 913)
	<<***extension***>> *(p. 116)* The maximum number of samples for which the built-in topic **dds::sub::Data↩ Reader** (p. 976) may store fragments at a given point in time.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_fragmented_samples** (p. 914)
	<<***extension***>> *(p. 116)* The initial number of samples for which a built-in topic **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragmented_samples_per_remote_writer** (p. 915)
	<<***extension***>> *(p. 116)* The maximum number of samples per remote writer for which a built-in topic **dds↩ ::sub::DataReader** (p. 976) may store fragments.

rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragments_per_sample** (p. 916)
	<<***extension***>> *(p. 116)* Maximum number of fragments for a single sample.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**dynamically_allocate_fragmented_samples** (p. 916)
	<<***extension***>> *(p. 116)* Determines whether the built-in topic **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.

6.61 Instance Lifecycle

QoS that control the lifecycle of instances.

QoS that control the lifecycle of instances.

Use these QoS fields to configure how RTI Connext manages instance lifecycles and resources in the DataReader and DataWriter queues.

Basic Settings

dds::core::policy::ResourceLimits (p. 2203)	**max_instances** (p. 2206)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Represents the maximum number of instances a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::ResourceLimits (p. 2203)	**initial_instances** (p. 2208)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Represents the initial number of instances a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) will manage.

Advanced Settings

dds::core::policy::DurabilityService (p. 1443)	**service_cleanup_delay** (p. 1445)
	<<***spec***>> *(p. 112)* Controls when the service is able to remove all information regarding a data instances.
dds::core::policy::DurabilityService (p. 1443)	**max_instances** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_samples_per_instance** (p. 1447)

	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::Reliability (p. 2149)	**instance_state_consistency_kind** (p. 2155)
	<<***extension***>> *(p. 116)* Whether instance state consistency is enabled
dds::core::policy::ResourceLimits (p. 2203)	**max_samples_per_instance** (p. 2207)
	<<***spec***>> *(p. 112)* Represents the maximum number of samples of any one instance a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::ResourceLimits (p. 2203)	**instance_hash_buckets** (p. 2208)
	[DEPRECATED] (p. **296)** <<***extension***>> *(p. 116)* Number of hash buckets for instances.
dds::core::policy::WriterDataLifecycle (p. 2660)	**autodispose_unregistered_instances** (p. 2661)
	<<***spec***>> *(p. 112)* Boolean flag that controls whether or not a writer will also dispose of an instance when the **dds::pub::DataWriter** (p. 1138) unregisters from the instance by means of the unregister operations.
dds::core::policy::WriterDataLifecycle (p. 2660)	**autopurge_unregistered_instances_delay** (p. 2662)
	<<***extension***>> *(p. 116)* Maximum duration for which the **dds::pub::DataWriter** (p. 1138) will maintain information regarding an instance once it has unregistered the instance.
dds::core::policy::WriterDataLifecycle (p. 2660)	**autopurge_disposed_instances_delay** (p. 2664)
	<<***extension***>> *(p. 116)* Maximum duration for which the **dds::pub::DataWriter** (p. 1138) will maintain information regarding an instance once it has disposed the instance.
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_nowriter_samples_delay** (p. 2140)
	<<***spec***>> *(p. 112)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain samples for an instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_↩ no_writers()** (p. 1617).
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_disposed_samples_delay** (p. 2141)
	<<***spec***>> *(p. 112)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain samples for an instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_↩ disposed()** (p. 1617).
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_disposed_instances_delay** (p. 2142)

	<<***extension***>> *(p. 116)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain information about a received instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) and there are no samples for the instance in the DataReader queue.
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_nowriter_instances_delay** (p. 2143)
	<<***extension***>> *(p. 116)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain information about a received instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) and there are no samples for the instance in the DataReader queue.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_total_instances** (p. 1096)
	<<***extension***>> *(p. 116)* Maximum number of instances for which a DataReader will keep state.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**keep_minimum_state_for_instances** (p. 1101)
	<<***extension***>> *(p. 116)* Whether or not keep a minimum instance state for up to **rti::core::policy::DataReaderResourceLimits::max_total_instances** (p. 1096).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**instance_replacement** (p. 1106)
	<<***extension***>> *(p. 116)* Sets the kind of instances allowed to be replaced for each instance state (**dds::sub::status::InstanceState** (p. 1615)) when a DataReader reaches **dds::core::policy::ResourceLimits::max_instances** (p. 2206).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**autopurge_remote_not_alive_writer_delay** (p. 1103)
	<<***extension***>> *(p. 116)* Maximum duration for which the **dds::sub::DataReader** (p. 976) will maintain information regarding a **dds::pub::DataWriter** (p. 1138) once the **dds::pub::DataWriter** (p. 1138) has become not alive.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**instance_replacement** (p. 1241)
	<<***extension***>> *(p. 116)* Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**replace_empty_instances** (p. 1242)
	<<***extension***>> *(p. 116)* Whether or not to replace empty instances during instance replacement

rti::core::policy::DataWriterResourceLimits (p. 1235)	**autoregister_instances** (p. 1242)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Whether or not to automatically register new instances
rti::core::policy::DataReaderProtocol (p. 1059)	**propagate_dispose_of_unregistered_instances** (p. 1063)
	<<***extension***>> *(p. 116)* Indicates whether or not an instance can move to the **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) state without being in the **dds::sub::status::InstanceState::alive()** (p. 1617) state.
rti::core::policy::DataReaderProtocol (p. 1059)	**propagate_unregister_of_disposed_instances** (p. 1064)
	<<***extension***>> *(p. 116)* Indicates whether or not an instance can move to the **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) state directly from the **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617).
rti::core::policy::DataWriterProtocol (p. 1210)	**disable_inline_keyhash** (p. 1214)
	<<***extension***>> *(p. 116)* Controls whether or not a keyhash is propagated on the wire with each sample.
rti::core::policy::DataWriterProtocol (p. 1210)	**serialize_key_with_dispose** (p. 1215)
	<<***extension***>> *(p. 116)* Controls whether or not the serialized key is propagated on the wire with dispose samples.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_writer_data_lifecycle** (p. 1291)
	<<***extension***>> *(p. 116)* Writer data lifecycle settings for a built-in publication writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_writer_data_lifecycle** (p. 1293)
	<<***extension***>> *(p. 116)* Writer data lifecycle settings for a built-in subscription writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_writer_data_lifecycle** (p. 1303)
	<<***extension***>> *(p. 116)* Writer data lifecycle settings for a built-in **rti::topic::ServiceRequest** (p. 2358) writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_writer_data_lifecycle** (p. 1316)
	<<***extension***>> *(p. 116)* Writer data lifecycle settings for a built-in participant configuration writer.

6.62 Sending Large Data

Use these QoS fields to optimize the process of sending large data.

Use these QoS fields to optimize the process of sending large data.

This includes QoS which help to reduce the number of copies needed and manage fragmentation.

See `Sending Large Data`, in the *Core Libraries User's Manual*, for more information.

Basic Settings

dds::core::policy::DataRepresentation (p. 1113)	**compression_settings** (p. 1117)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Structure that contains the compression settings.

Advanced Settings

rti::core::CompressionSettings (p. 943)	**compression_ids** (p. 945)
	<<***extension***>> *(p. 116)* Mask that represents the compression algorithms enabled.
rti::core::CompressionSettings (p. 943)	**writer_compression_level** (p. 946)
	<<***extension***>> *(p. 116)* The level of compression to use when compressing data.
rti::core::CompressionSettings (p. 943)	**writer_compression_threshold** (p. 947)
	<<***extension***>> *(p. 116)* The threshold, in bytes, above which a serialized sample will be eligible to be compressed.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**disable_fragmentation_support** (p. 1092)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) can receive fragmented samples.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragmented_samples** (p. 1092)
	<<***extension***>> *(p. 116)* The maximum number of samples for which the **dds::sub::DataReader** (p. 976) may store fragments at a given point in time.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_fragmented_samples** (p. 1093)

	<<***extension***>> *(p. 116)* The initial number of samples for which a **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragmented_samples_per_remote_writer** (p. 1094)
	<<***extension***>> *(p. 116)* The maximum number of samples per remote writer for which a **dds::sub::Data↩ Reader** (p. 976) may store fragments.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragments_per_sample** (p. 1094)
	<<***extension***>> *(p. 116)* Maximum number of fragments for a single sample.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**dynamically_allocate_fragmented_samples** (p. 1095)
	<<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**shmem_ref_transfer_mode_attached_segment_↩ allocation** (p. 1106)
	<<***extension***>> *(p. 116)* Allocation resource for the shared memory segments attached by the **dds::sub↩ ::DataReader** (p. 976).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**writer_loaned_sample_allocation** (p. 1247)
	<<***extension***>> *(p. 116)* Represents the allocation settings of loaned samples managed by a **dds::pub↩ ::DataWriter** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initialize_writer_loaned_sample** (p. 1248)
	<<***extension***>> *(p. 116)* Whether or not to initialize loaned samples returned by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterTransferMode (p. 1253)	**shmem_ref_settings** (p. 1254)
	<<***extension***>> *(p. 116)* Settings related to transferring data using shared memory references.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**shmem_ref_transfer_mode_max_segments** (p. 1433)
	<<***extension***>> *(p. 116)* Maximum number of segments created by all DataWriters belonging to a **dds↩ ::domain::DomainParticipant** (p. 1326).
rti::core::policy::DiscoveryConfig (p. 1271)	**endpoint_type_object_lb_serialization_threshold** (p. 1310)

	<<***extension***>> *(*p. *116)* Controls when/if the TypeObject v1 is compressed when propagated in Simple Endpoint Discovery.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**disable_fragmentation_support** (p. 913)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Determines whether the built-in topic **dds::sub::DataReader** (p. 976) can receive fragmented samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragmented_samples** (p. 913)
	<<***extension***>> *(*p. *116)* The maximum number of samples for which the built-in topic **dds::sub::DataReader** (p. 976) may store fragments at a given point in time.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_fragmented_samples** (p. 914)
	<<***extension***>> *(*p. *116)* The initial number of samples for which a built-in topic **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragmented_samples_per_remote_writer** (p. 915)
	<<***extension***>> *(*p. *116)* The maximum number of samples per remote writer for which a built-in topic **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragments_per_sample** (p. 916)
	<<***extension***>> *(*p. *116)* Maximum number of fragments for a single sample.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**dynamically_allocate_fragmented_samples** (p. 916)
	<<***extension***>> *(*p. *116)* Determines whether the built-in topic **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.

6.63 Liveliness

Use these QoS fields to manage the liveliness of participants and endpoints, ensuring active communication and timely updates of the system state.

Use these QoS fields to manage the liveliness of participants and endpoints, ensuring active communication and timely updates of the system state.

Basic Settings

dds::core::policy::Liveliness (p. 1645)	**kind** (p. 1649)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of liveliness desired.
dds::core::policy::Liveliness (p. 1645)	**lease_duration** (p. 1649)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The duration within which a **dds::pub::DataWriter** (p. 1138) must be asserted, or else it is assumed to be not alive.
dds::core::policy::Liveliness (p. 1645)	**assertions_per_lease_duration** (p. 1651)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The number of assertions a **dds::pub::DataWriter** (p. 1138) will send during a its **dds::core::policy::Liveliness::lease_duration** (p. 1649).
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_liveliness_lease_duration** (p. 1279)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The liveliness lease duration for the participant.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_liveliness_assert_period** (p. 1279)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The period to assert liveliness for the participant.

Advanced Settings

rti::core::policy::DataReaderResourceLimits (p. 1081)	**autopurge_remote_not_alive_writer_delay** (p. 1103)
	*<<**extension**>> (p. 116)* Maximum duration for which the **dds::sub::DataReader** (p. 976) will maintain information regarding a **dds::pub::DataWriter** (p. 1138) once the **dds::pub::DataWriter** (p. 1138) has become not alive.
rti::core::policy::DiscoveryConfig (p. 1271)	**remote_participant_purge_kind** (p. 1281)
	[DEPRECATED] (p. 296) *<<**extension**>> (p. 116)* The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
rti::core::policy::DiscoveryConfig (p. 1271)	**max_liveliness_loss_detection_period** (p. 1281)
	[DEPRECATED] (p. 296) *<<**extension**>> (p. 116)* The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_writer** (p. 1296)

	<<***extension***>> *(*p. *116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **dds::core::policy::↩ ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core↩ ::policy::ReliabilityKind_def** (p. 2156).
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_reader_reliability_kind** (p. 1295)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Reliability policy for a built-in participant message reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_reader** (p. 1295)
	<<***extension***>> *(*p. *116)* RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **rti::core::policy::DiscoveryConfig::participant_↩ message_reader_reliability_kind** (p. 1295) is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157).
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_heartbeat_retries** (p. 2239)
	<<***extension***>> *(*p. *116)* The maximum number of *periodic* heartbeat retries before marking a remote reader as inactive.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**inactivate_nonprogressing_readers** (p. 2239)
	<<***extension***>> *(*p. *116)* Whether to treat remote readers as inactive when their NACKs do not progress

6.64 Logging

Use these QoS fields to configure the RTI Connext logging infrastructure.

Use these QoS fields to configure the RTI Connext logging infrastructure.

This includes settings for configuring the verbosity and directing the output.

See `Logging`, in the *Core Libraries User's Manual*, for more information.

Basic Settings

rti::config::Logger (p. 1683)	**verbosity** (p. 1685)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The verbosities at which RTI Connext diagnostic information is logged.
rti::config::Logger (p. 1683)	**category** (p. 1685)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Categories of logged messages.
rti::config::Logger (p. 1683)	**output_file** (p. 1686)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Specifies the file to which log messages will be redirected to.
rti::config::Logger (p. 1683)	**output_file_suffix** (p. 1687)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sets the file suffix when logging to a set of files
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**max_historical_logs** (p. 1729)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**publication_period** (p. 1732)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Period at which the logging thread publishes log messages.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**middleware_forwarding_level** (p. 1734)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Log messages with **rti::config::LogFacility::middleware** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**security_event_forwarding_level** (p. 1734)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Log messages with **rti::config::LogFacility::security_event** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**service_forwarding_level** (p. 1735)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Log messages with **rti::config::LogFacility::service** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**user_forwarding_level** (p. 1736)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Log messages with **rti::config::LogFacility::user** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringTelemetryData (p. 1744)	**logs** (p. 1746)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* **rti::core::MonitoringLoggingForwardingSettings** (p. 1733) containing the **rti::config::SyslogVerbosity** (p. 458) levels that will be forwarded for the different **rti::config::LogFacility** (p. 459).

Advanced Settings

rti::config::Logger (p. 1683)	**print_format** (p. 1688)
	<<***extension***>> *(p. 116)* The format used to output RTI Connext diagnostic information.
rti::config::Logger (p. 1683)	**max_bytes_per_file** (p. 1687)
	<<***extension***>> *(p. 116)* Specifies the maximum number of bytes a single file can contain
rti::config::Logger (p. 1683)	**max_files** (p. 1687)
	<<***extension***>> *(p. 116)* Specifies the maximum number of files to create before overwritting the previous ones.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**concurrency_level** (p. 1729)
	<<***extension***>> *(p. 116)* Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**datawriter_qos_profile_name** (p. 1730)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes log messages.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**thread** (p. 1731)
	<<***extension***>> *(p. 116)* The settings of the logging thread.
rti::core::MonitoringDistributionSettings (p. 1716)	**logging_settings** (p. 1721)
	<<***extension***>> *(p. 116)* Configures the distribution of logging messages.

6.65 Middleware Memory Management

Use these QoS fields to configure the memory usage of the middleware, including various initial and max limits.

Use these QoS fields to configure the memory usage of the middleware, including various initial and max limits.

Basic Settings

dds::core::policy::ResourceLimits (p. 2203)	**max_samples** (p. 2205)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Represents the maximum samples the middleware can store for any one **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)).
dds::core::policy::ResourceLimits (p. 2203)	**max_instances** (p. 2206)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Represents the maximum number of instances a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::ResourceLimits (p. 2203)	**initial_samples** (p. 2207)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Represents the initial samples the middleware will store for any one **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)).
dds::core::policy::ResourceLimits (p. 2203)	**initial_instances** (p. 2208)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Represents the initial number of instances a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) will manage.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**max_historical_logs** (p. 1729)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.

Advanced Settings

rti::core::PersistentStorageSettings (p. 1927)	**writer_memory_state** (p. 1936)

	<<***extension***>> *(p. 116)* Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.
dds::core::policy::DurabilityService (p. 1443)	**max_samples** (p. 1446)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_instances** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_samples_per_instance** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::ResourceLimits (p. 2203)	**max_samples_per_instance** (p. 2207)
	<<***spec***>> *(p. 112)* Represents the maximum number of samples of any one instance a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_writers** (p. 1085)
	<<***extension***>> *(p. 116)* The maximum number of remote writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_writers_per_instance** (p. 1086)
	<<***extension***>> *(p. 116)* The maximum number of remote writers from which a **dds::sub::DataReader** (p. 976) may read a single instance.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_samples_per_remote_writer** (p. 1087)
	<<***extension***>> *(p. 116)* The maximum number of out-of-order samples from a given remote **dds::pub::Data↩ Writer** (p. 1138) that a **dds::sub::DataReader** (p. 976) may store when maintaining a reliable connection to the **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_infos** (p. 1087)
	<<***extension***>> *(p. 116)* The maximum number of info units that a **dds::sub::DataReader** (p. 976) can use to store **dds::sub::SampleInfo** (p. 2285).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_remote_writers** (p. 1088)

	<<***extension***>> *(p. 116)* The initial number of remote writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_remote_writers_per_instance** (p. 1089)
	<<***extension***>> *(p. 116)* The initial number of remote writers from which a **dds::sub::DataReader** (p. 976) may read a single instance.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_infos** (p. 1089)
	<<***extension***>> *(p. 116)* The initial number of info units that a **dds::sub::DataReader** (p. 976) can have, which are used to store **dds::sub::SampleInfo** (p. 2285).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_outstanding_reads** (p. 1090)
	<<***extension***>> *(p. 116)* The initial number of outstanding calls to read/take (or one of their variants) on the same **dds::sub::DataReader** (p. 976) for which memory has not been returned by calling **dds::sub::LoanedSamples::return_loan()** (p. 1667).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_outstanding_reads** (p. 1091)
	<<***extension***>> *(p. 116)* The maximum number of outstanding read/take calls (or one of their variants) on the same **dds::sub::DataReader** (p. 976) for which memory has not been returned by calling **dds::sub::LoanedSamples::return_loan()** (p. 1667).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_samples_per_read** (p. 1091)
	<<***extension***>> *(p. 116)* The maximum number of data samples that the application can receive from the middleware in a single call to **dds::sub::DataReader::read** (p. 996) or **dds::sub::DataReader::take** (p. 997). If more data exists in the middleware, the application will need to issue multiple read/take calls.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragmented_samples** (p. 1092)
	<<***extension***>> *(p. 116)* The maximum number of samples for which the **dds::sub::DataReader** (p. 976) may store fragments at a given point in time.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_fragmented_samples** (p. 1093)

	<<***extension***>> *(p. 116)* The initial number of samples for which a **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragmented_samples_per_remote_writer** (p. 1094)
	<<***extension***>> *(p. 116)* The maximum number of samples per remote writer for which a **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragments_per_sample** (p. 1094)
	<<***extension***>> *(p. 116)* Maximum number of fragments for a single sample.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**dynamically_allocate_fragmented_samples** (p. 1095)
	<<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_total_instances** (p. 1096)
	<<***extension***>> *(p. 116)* Maximum number of instances for which a DataReader will keep state.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_virtual_writers** (p. 1097)
	<<***extension***>> *(p. 116)* The maximum number of remote virtual writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_remote_virtual_writers** (p. 1097)
	<<***extension***>> *(p. 116)* The initial number of remote virtual writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_virtual_writers_per_instance** (p. 1098)
	<<***extension***>> *(p. 116)* The maximum number of virtual remote writers that can be associated with an instance.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_remote_virtual_writers_per_instance** (p. 1099)
	<<***extension***>> *(p. 116)* The initial number of virtual remote writers per instance.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_writers_per_sample** (p. 1099)

	<<***extension***>> *(*p. *116)* The maximum number of remote writers allowed to write the same sample.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_query_condition_filters** (p. 1100)
	<<***extension***>> *(*p. *116)* The maximum number of query condition filters a reader is allowed.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_app_ack_response_length** (p. 1101)
	<<***extension***>> *(*p. *116)* Maximum length of application-level acknowledgment response data
rti::core::policy::DataReaderResourceLimits (p. 1081)	**keep_minimum_state_for_instances** (p. 1101)
	<<***extension***>> *(*p. *116)* Whether or not keep a minimum instance state for up to **rti::core::policy::DataReaderResourceLimits::max_total_instances** (p. 1096).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_topic_queries** (p. 1102)
	<<***extension***>> *(*p. *116)* The initial number of TopicQueries allocated by a **dds::sub::DataReader** (p. 976).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_topic_queries** (p. 1103)
	<<***extension***>> *(*p. *116)* The maximum number of active TopicQueries that a **dds::sub::DataReader** (p. 976) can create.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**shmem_ref_transfer_mode_attached_segment_allocation** (p. 1106)
	<<***extension***>> *(*p. *116)* Allocation resource for the shared memory segments attached by the **dds::sub::DataReader** (p. 976).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**instance_replacement** (p. 1106)
	<<***extension***>> *(*p. *116)* Sets the kind of instances allowed to be replaced for each instance state (**dds::sub::status::InstanceState** (p. 1615)) when a DataReader reaches **dds::core::policy::ResourceLimits::max_instances** (p. 2206).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**autopurge_remote_not_alive_writer_delay** (p. 1103)

	<<***extension***>> *(*p. *116)* Maximum duration for which the **dds::sub::DataReader** (p. 976) will maintain information regarding a **dds::pub::DataWriter** (p. 1138) once the **dds::pub::DataWriter** (p. 1138) has become not alive.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**lazily_allocate_batch_resources** (p. 1105)
	<<***extension***>> *(*p. *116)* Determines whether the **dds::sub::DataReader** (p. 976) pre-allocates resources for receiving batches of samples or lazily allocates them when needed.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_concurrent_blocking_threads** (p. 1238)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* The initial number of threads that are allowed to concurrently block on write call on the same **dds::pub::Data↩ Writer** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_concurrent_blocking_threads** (p. 1238)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* The maximum number of threads that are allowed to concurrently block on write call on the same **dds::pub::↩ DataWriter** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_remote_reader_filters** (p. 1239)
	<<***extension***>> *(*p. *116)* The maximum number of remote DataReaders for which the **dds::pub::DataWriter** (p. 1138) will perform content-based filtering.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_batches** (p. 1239)
	<<***extension***>> *(*p. *116)* Represents the initial number of batches a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_batches** (p. 1240)
	<<***extension***>> *(*p. *116)* Represents the maximum number of batches a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**instance_replacement** (p. 1241)
	<<***extension***>> *(*p. *116)* Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**replace_empty_instances** (p. 1242)
	<<***extension***>> *(*p. *116)* Whether or not to replace empty instances during instance replacement
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_virtual_writers** (p. 1244)

	<<***extension***>> *(p. 116)* The maximum number of virtual writers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_virtual_writers** (p. 1243)
	<<***extension***>> *(p. 116)* The initial number of virtual writers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_remote_readers** (p. 1244)
	<<***extension***>> *(p. 116)* Controls the initial memory allocation for the matched remote readers of a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_remote_readers** (p. 1244)
	<<***extension***>> *(p. 116)* The maximum number of remote readers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_app_ack_remote_readers** (p. 1245)
	<<***extension***>> *(p. 116)* The maximum number of application-level acknowledging remote readers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_active_topic_queries** (p. 1246)
	<<***extension***>> *(p. 116)* Represents the initial number of active topic queries a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_active_topic_queries** (p. 1246)
	<<***extension***>> *(p. 116)* Represents the maximum number of active topic queries a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**writer_loaned_sample_allocation** (p. 1247)
	<<***extension***>> *(p. 116)* Represents the allocation settings of loaned samples managed by a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::SystemResourceLimits (p. 2443)	**max_objects_per_thread** (p. 2445)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* The maximum number of objects that can be stored per thread for a DomainParticipantFactory.
rti::core::policy::SystemResourceLimits (p. 2443)	**initial_objects_per_thread** (p. 2445)
	<<***extension***>> *(p. 116)* The number of objects per thread for a DomainParticipantFactory for which infrastructure will initially be allocated.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_writer_allocation** (p. 1398)

	<<***extension***>> *(p. 116)* Allocation settings applied to local DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_reader_allocation** (p. 1398)
	<<***extension***>> *(p. 116)* Allocation settings applied to local DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_publisher_allocation** (p. 1399)
	<<***extension***>> *(p. 116)* Allocation settings applied to local Publisher.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_subscriber_allocation** (p. 1399)
	<<***extension***>> *(p. 116)* Allocation settings applied to local Subscriber.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_topic_allocation** (p. 1399)
	<<***extension***>> *(p. 116)* Allocation settings applied to local Topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_writer_allocation** (p. 1400)
	<<***extension***>> *(p. 116)* Allocation settings applied to remote DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_reader_allocation** (p. 1401)
	<<***extension***>> *(p. 116)* Allocation settings applied to remote DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_participant_allocation** (p. 1401)
	<<***extension***>> *(p. 116)* Allocation settings applied to remote DomainParticipants.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_writer_reader_pair_allocation** (p. 1402)
	<<***extension***>> *(p. 116)* Allocation settings applied to matching local writer and remote/local reader pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_reader_writer_pair_allocation** (p. 1402)
	<<***extension***>> *(p. 116)* Allocation settings applied to matching local reader and remote/local writer pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**ignored_entity_allocation** (p. 1403)

	<<***extension***>> *(p. 116)* Allocation settings applied to ignored entities.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filtered_topic_allocation** (p. 1403)
	<<***extension***>> *(p. 116)* Allocation settings applied to content filtered topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filter_allocation** (p. 1404)
	<<***extension***>> *(p. 116)* Allocation settings applied to content filter.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**read_condition_allocation** (p. 1404)
	<<***extension***>> *(p. 116)* Allocation settings applied to read condition pool.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**query_condition_allocation** (p. 1405)
	<<***extension***>> *(p. 116)* Allocation settings applied to query condition pool.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**outstanding_asynchronous_sample_allocation** (p. 1405)
	<<***extension***>> *(p. 116)* Allocation settings applied to the maximum number of samples (from all **dds::pub::↩ DataWriter** (p. 1138)) waiting to be asynchronously written.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**flow_controller_allocation** (p. 1406)
	<<***extension***>> *(p. 116)* Allocation settings applied to flow controllers.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_writer_hash_buckets** (p. 1406)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Hash_Buckets settings applied to local DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_reader_hash_buckets** (p. 1407)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_publisher_hash_buckets** (p. 1407)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local Publisher.

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_subscriber_hash_buckets** (p. 1408)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local Subscriber.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_topic_hash_buckets** (p. 1408)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local Topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_writer_hash_buckets** (p. 1409)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_reader_hash_buckets** (p. 1409)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_participant_hash_buckets** (p. 1410)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote DomainParticipants.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_writer_reader_pair_hash_buckets** (p. 1410)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for matching local writer and remote/local reader pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_reader_writer_pair_hash_buckets** (p. 1411)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for matching local reader and remote/local writer pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**ignored_entity_hash_buckets** (p. 1411)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for ignored entities.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filtered_topic_hash_buckets** (p. 1412)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for content filtered topics.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filter_hash_buckets** (p. 1412)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for content filters.

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**flow_controller_hash_buckets** (p. 1413)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for flow controllers.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_gather_destinations** (p. 1413)
	<<***extension***>> *(p. 116)* Maximum number of destinations per RTI Connext send.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**participant_user_data_max_length** (p. 1414)
	<<***extension***>> *(p. 116)* Maximum length of user data in **dds::domain::qos::DomainParticipantQos** (p. 1384) and **dds::topic::ParticipantBuiltinTopicData** (p. 1908)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**topic_data_max_length** (p. 1414)
	<<***extension***>> *(p. 116)* Maximum length of topic data in **dds::topic::qos::TopicQos** (p. 2509), **dds::topic::TopicBuiltinTopicData** (p. 2493), **dds::topic::PublicationBuiltinTopicData** (p. 1977) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**publisher_group_data_max_length** (p. 1415)
	<<***extension***>> *(p. 116)* Maximum length of group data in **dds::pub::qos::PublisherQos** (p. 2007) and **dds::topic::PublicationBuiltinTopicData** (p. 1977)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**subscriber_group_data_max_length** (p. 1415)
	<<***extension***>> *(p. 116)* Maximum length of group data in **dds::sub::qos::SubscriberQos** (p. 2423) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_user_data_max_length** (p. 1416)
	<<***extension***>> *(p. 116)* Maximum length of user data in **dds::pub::qos::DataWriterQos** (p. 1226) and **dds::topic::PublicationBuiltinTopicData** (p. 1977)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_user_data_max_length** (p. 1416)
	<<***extension***>> *(p. 116)* Maximum length of user data in **dds::sub::qos::DataReaderQos** (p. 1072) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_partitions** (p. 1417)
	<<***extension***>> *(p. 116)* Maximum number of partition name strings allowable in a **dds::core::policy::Partition** (p. 1923).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_partition_cumulative_characters** (p. 1417)
	<<***extension***>> *(p. 116)* Maximum number of combined characters allowable in all partition names in a **dds::core::policy::Partition** (p. 1923).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_code_max_serialized_length** (p. 1418)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Maximum size of serialized string for type code.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_object_max_serialized_length** (p. 1419)
	<<***extension***>> *(p. 116)* The maximum length, in bytes, that the buffer to serialize a TypeObject v1 can consume.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_object_max_deserialized_length** (p. 1420)
	<<***extension***>> *(p. 116)* The maximum number of bytes that a deserialized TypeObject v1 can consume.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**deserialized_type_object_dynamic_allocation_threshold** (p. 1420)
	<<***extension***>> *(p. 116)* A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject v1.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**serialized_type_object_dynamic_allocation_threshold** (p. 1421)
	<<***extension***>> *(p. 116)* A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject v1.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**contentfilter_property_max_length** (p. 1422)
	<<***extension***>> *(p. 116)* This field is the maximum length of all data related to a Content-filtered topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**channel_seq_max_length** (p. 1423)
	<<***extension***>> *(p. 116)* Maximum number of channels that can be specified in **rti::core::policy::MultiChannel** (p. 1749) for MultiChannel DataWriters.

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**channel_filter_expression_max_length** (p. 1424)
	<<***extension***>> *(p. 116)* Maximum length of a channel **rti::core::ChannelSettings::filter_expression** (p. 923) in a MultiChannel DataWriter.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**participant_property_list_max_length** (p. 1424)
	<<***extension***>> *(p. 116)* Maximum number of properties associated with the **dds::domain::DomainParticipant** (p. 1326).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**participant_property_string_max_length** (p. 1425)
	<<***extension***>> *(p. 116)* Maximum string length of the properties associated with the **dds::domain::DomainParticipant** (p. 1326).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_property_list_max_length** (p. 1425)
	<<***extension***>> *(p. 116)* Maximum number of properties associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_property_string_max_length** (p. 1426)
	<<***extension***>> *(p. 116)* Maximum string length of the properties associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_property_list_max_length** (p. 1426)
	<<***extension***>> *(p. 116)* Maximum number of properties associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_property_string_max_length** (p. 1427)
	<<***extension***>> *(p. 116)* Maximum string length of the properties associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_endpoint_groups** (p. 1427)
	<<***extension***>> *(p. 116)* Maximum number of **rti::core::EndpointGroup** (p. 1515) allowable in a **rti::core::policy::Availability** (p. 868).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_endpoint_group_cumulative_characters** (p. 1428)

	<<***extension***>> *(*p. *116)* Maximum number of combined role_name characters allowed in all **rti::core↩ ::EndpointGroup** (p. 1515) in a **rti::core::policy::↩ Availability** (p. 868).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**transport_info_list_max_length** (p. 1428)
	<<***extension***>> *(*p. *116)* Maximum number of installed transports to send and receive information about in **dds::topic::ParticipantBuiltinTopic↩ Data::transport_info** (p. 1914).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**ignored_entity_replacement_kind** (p. 1429)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **rti::core::policy::DomainParticipant↩ ResourceLimits::ignored_entity_allocation** (p. 1403) are reached.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_topic_query_allocation** (p. 1429)
	<<***extension***>> *(*p. *116)* Allocation settings applied to remote TopicQueries.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_topic_query_hash_buckets** (p. 1430)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Number of hash buckets for remote TopicQueries.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_data_tag_list_max_length** (p. 1431)
	<<***extension***>> *(*p. *116)* Maximum number of data tags associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_data_tag_string_max_length** (p. 1431)
	<<***extension***>> *(*p. *116)* Maximum string length of the data tags associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_data_tag_list_max_length** (p. 1432)
	<<***extension***>> *(*p. *116)* Maximum number of data tags associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_data_tag_string_max_length** (p. 1432)

	<<***extension***>> *(*p. *116)* Maximum string length of the data tags associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**shmem_ref_transfer_mode_max_segments** (p. 1433)
	<<***extension***>> *(*p. *116)* Maximum number of segments created by all DataWriters belonging to a **dds↩ ::domain::DomainParticipant** (p. 1326).
rti::core::policy::Event (p. 1541)	**initial_count** (p. 1543)
	<<***extension***>> *(*p. *116)* The initial number of events.
rti::core::policy::Event (p. 1541)	**max_count** (p. 1543)
	<<***extension***>> *(*p. *116)* The maximum number of events.
rti::core::policy::Database (p. 969)	**initial_records** (p. 974)
	<<***extension***>> *(*p. *116)* The initial number of total records.
rti::core::policy::Database (p. 969)	**max_skiplist_level** (p. 974)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* The maximum level of the skiplist.
rti::core::policy::Database (p. 969)	**initial_weak_references** (p. 975)
	<<***extension***>> *(*p. *116)* The initial number of weak references.
rti::core::policy::Database (p. 969)	**max_weak_references** (p. 974)
	<<***extension***>> *(*p. *116)* The maximum number of weak references.
rti::core::policy::ReceiverPool (p. 2144)	**buffer_size** (p. 2147)
	<<***extension***>> *(*p. *116)* The receive buffer size in bytes.
rti::core::policy::ReceiverPool (p. 2144)	**buffer_alignment** (p. 2148)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* The receive buffer alignment.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_reader_resource_limits** (p. 1285)
	<<***extension***>> *(*p. *116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_reader_resource_limits** (p. 1287)
	<<***extension***>> *(*p. *116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_reader_resource_limits** (p. 1289)
	<<***extension***>> *(*p. *116)* Resource limits.

rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_reader_resource_limits** (p. 1313)
	<<***extension***>> *(p. 116)* Resource limits for the built-in topic participant configuration reader.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_samples** (p. 909)
	<<***extension***>> *(p. 116)* Initial number of samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_samples** (p. 909)
	<<***extension***>> *(p. 116)* Maximum number of samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_infos** (p. 910)
	<<***extension***>> *(p. 116)* Initial number of sample infos.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_infos** (p. 911)
	<<***extension***>> *(p. 116)* Maximum number of sample infos.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_outstanding_reads** (p. 911)
	<<***extension***>> *(p. 116)* The initial number of outstanding reads that have not called finish yet on the same built-in topic **dds::sub::DataReader** (p. 976).
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_outstanding_reads** (p. 912)
	<<***extension***>> *(p. 116)* The maximum number of outstanding reads that have not called finish yet on the same built-in topic **dds::sub::DataReader** (p. 976).
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_samples_per_read** (p. 912)
	<<***extension***>> *(p. 116)* Maximum number of samples that can be read/taken on a same built-in topic **dds::sub::DataReader** (p. 976).
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**disable_fragmentation_support** (p. 913)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Determines whether the built-in topic **dds::sub::Data↩ Reader** (p. 976) can receive fragmented samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragmented_samples** (p. 913)

	<<***extension***>> *(p. 116)* The maximum number of samples for which the built-in topic **dds::sub::DataReader** (p. 976) may store fragments at a given point in time.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_fragmented_samples** (p. 914)
	<<***extension***>> *(p. 116)* The initial number of samples for which a built-in topic **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragmented_samples_per_remote_writer** (p. 915)
	<<***extension***>> *(p. 116)* The maximum number of samples per remote writer for which a built-in topic **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragments_per_sample** (p. 916)
	<<***extension***>> *(p. 116)* Maximum number of fragments for a single sample.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**dynamically_allocate_fragmented_samples** (p. 916)
	<<***extension***>> *(p. 116)* Determines whether the built-in topic **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.
rti::core::ThreadSettings (p. 2449)	**stack_size** (p. 2451)
	The thread stack-size.
rti::config::Logger (p. 1683)	**max_bytes_per_file** (p. 1687)
	<<***extension***>> *(p. 116)* Specifies the maximum number of bytes a single file can contain
rti::config::Logger (p. 1683)	**max_files** (p. 1687)
	<<***extension***>> *(p. 116)* Specifies the maximum number of files to create before overwritting the previous ones.

6.66 QoS Profile Management

Use these QoS fields to configure how QoS profiles are loaded and which ones should be used when entities are created.

Use these QoS fields to configure how QoS profiles are loaded and which ones should be used when entities are created.

Basic Settings

rti::core::QosProviderParams (p. 2049)	**string_profile** (p. 2050)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sequence of strings containing a XML document to load.
rti::core::QosProviderParams (p. 2049)	**url_profile** (p. 2050)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sequence of **URL groups** (p. 304) containing a set of XML documents to load.
rti::core::QosProviderParams (p. 2049)	**ignore_user_profile** (p. 2051)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Ignores the file USER_QOS_PROFILES.xml in the current working directory.
rti::core::QosProviderParams (p. 2049)	**ignore_environment_profile** (p. 2052)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Ignores the value of the **NDDS_QOS_PROFILES** (p. 304).
rti::core::QosProviderParams (p. 2049)	**ignore_resource_profile** (p. 2052)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Ignores the file NDDS_QOS_PROFILES.xml.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**enable** (p. 1713)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Enables the use of a dedicated **dds::domain::Domain↩ Participant** (p. 1326) to distribute the RTI Connext application telemetry data.

Advanced Settings

rti::core::MonitoringEventDistributionSettings (p. 1722)	**datawriter_qos_profile_name** (p. 1724)
	*<<**extension**>> (p. 116)* The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes event metrics.
rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**datawriter_qos_profile_name** (p. 1742)
	*<<**extension**>> (p. 116)* The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes periodic metrics.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**datawriter_qos_profile_name** (p. 1730)
	*<<**extension**>> (p. 116)* The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes log messages.

rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**participant_qos_profile_name** (p. 1714)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the **dds::domain::DomainParticipant** (p. 1326) that will be used to distribute telemetry data.
rti::core::MonitoringDistributionSettings (p. 1716)	**dedicated_participant** (p. 1719)
	<<***extension***>> *(p. 116)* Configures the use of a dedicated **dds::domain::DomainParticipant** (p. 1326) to distribute the RTI Connext application telemetry data.
rti::core::MonitoringDistributionSettings (p. 1716)	**publisher_qos_profile_name** (p. 1718)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.

6.67 Reliability QoS

Reliability related QoS.

Reliability related QoS.

Use these QoS fields to configure the reliable delivery of your application data – this includes those settings that configure the reliability protocol.

See `Reliable Delivery Model`, in the *Core Libraries User's Manual*, for more information about the DDS reliability protocol.

Basic Settings

dds::core::policy::Reliability (p. 2149)	**kind** (p. 2153)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Kind of reliability.
dds::core::policy::Reliability (p. 2149)	**max_blocking_time** (p. 2153)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* The maximum time a DataWriter can block on a **write()** (p. 1179) call.
dds::core::policy::History (p. 1602)	**kind** (p. 1604)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Specifies the kind of history to be kept.
dds::core::policy::History (p. 1602)	**depth** (p. 1605)

	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Specifies the number of samples per instance to be kept, when the `kind` is **dds::core::policy::HistoryKind_def::KEEP↩ _LAST** (p. 1607)
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**heartbeat_period** (p. 2234)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The period at which to send heartbeats.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**fast_heartbeat_period** (p. 2235)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**late_joiner_heartbeat_period** (p. 2236)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**min_send_window_size** (p. 2249)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Minimum size of send window of unacknowledged samples
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_send_window_size** (p. 2250)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Maximum size of send window of unacknowledged samples

Advanced Settings

rti::core::policy::Availability (p. 868)	**enable_required_subscriptions** (p. 871)
	<<***extension***>> *(p. 116)* Enables support for required subscriptions in a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::Availability (p. 868)	**max_data_availability_waiting_time** (p. 871)
	<<***extension***>> *(p. 116)* Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
rti::core::policy::Availability (p. 868)	**max_endpoint_availability_waiting_time** (p. 872)

	<<***extension***>> *(p. 116)* Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
rti::core::policy::Availability (p. 868)	**required_matched_endpoint_groups** (p. 873)
	<<***extension***>> *(p. 116)* A sequence of endpoint groups.
dds::core::policy::Reliability (p. 2149)	**acknowledgment_kind** (p. 2154)
	<<***extension***>> *(p. 116)* Kind of reliable acknowledgment
dds::core::policy::Reliability (p. 2149)	**instance_state_consistency_kind** (p. 2155)
	<<***extension***>> *(p. 116)* Whether instance state consistency is enabled
dds::core::policy::Reliability (p. 2149)	**exact_reliability_kind_match** (p. 2155)
	<<***extension***>> *(p. 116)* Whether an exact reliability kind match is required for endpoint matching.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_app_ack_response_length** (p. 1101)
	<<***extension***>> *(p. 116)* Maximum length of application-level acknowledgment response data
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_app_ack_remote_readers** (p. 1245)
	<<***extension***>> *(p. 116)* The maximum number of application-level acknowledging remote readers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataReaderProtocol (p. 1059)	**rtps_reliable_reader** (p. 1064)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **dds::sub::DataReader** (p. 976). This parameter only has effect if the reader is configured with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core::policy::ReliabilityKind_def** (p. 2156).
rti::core::policy::DataReaderProtocol (p. 1059)	**disable_positive_acks** (p. 1062)
	<<***extension***>> *(p. 116)* Whether the reader sends positive acknowledgements to writers.
rti::core::policy::DataWriterProtocol (p. 1210)	**push_on_write** (p. 1213)
	<<***extension***>> *(p. 116)* Whether to push sample out when write is called.
rti::core::policy::DataWriterProtocol (p. 1210)	**disable_positive_acks** (p. 1214)
	<<***extension***>> *(p. 116)* Controls whether or not the writer expects positive acknowledgements from matching readers.

rti::core::policy::DataWriterProtocol (p. 1210)	**propagate_app_ack_with_no_response** (p. 1216)
	<<***extension***>> *(p. 116)* Controls whether or not a **dds::pub::DataWriter** (p. 1138) receives **dds::pub::DataWriterListener::on_application_acknowledgment** (p. 1207) notifications with an empty or invalid response.
rti::core::policy::DataWriterProtocol (p. 1210)	**rtps_reliable_writer** (p. 1217)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **dds::pub::DataWriter** (p. 1138). This parameter only has effect if both the writer and the matching reader are configured with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core::policy::ReliabilityKind_def** (p. 2156).
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_reader** (p. 1286)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_reader** (p. 1288)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_writer** (p. 1296)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core::policy::ReliabilityKind_def** (p. 2156).
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_reader_reliability_kind** (p. 1295)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Reliability policy for a built-in participant message reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_reader** (p. 1295)
	<<***extension***>> *(p. 116)* RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **rti::core::policy::DiscoveryConfig::participant_message_reader_reliability_kind** (p. 1295) is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157).
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_writer** (p. 1289)

	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_writer** (p. 1291)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_writer** (p. 1301)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **rti::topic::ServiceRequest** (p. 2358) writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_reader** (p. 1304)
	<<***extension***>> *(p. 116)* RTPS reliable reader protocol-related configuration settings for a built-in **rti::topic::ServiceRequest** (p. 2358) reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_writer** (p. 1307)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_reader** (p. 1309)
	<<***extension***>> *(p. 116)* RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_reader** (p. 1312)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_writer** (p. 1314)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
rti::core::RtpsReliableReaderProtocol (p. 2221)	**min_heartbeat_response_delay** (p. 2223)
	<<***extension***>> *(p. 116)* The minimum delay to respond to a heartbeat.
rti::core::RtpsReliableReaderProtocol (p. 2221)	**max_heartbeat_response_delay** (p. 2224)
	<<***extension***>> *(p. 116)* The maximum delay to respond to a heartbeat.
rti::core::RtpsReliableReaderProtocol (p. 2221)	**heartbeat_suppression_duration** (p. 2224)

	<<***extension***>> *(p. 116)* The duration a reader ignores consecutively received heartbeats
rti::core::RtpsReliableReaderProtocol (p. 2221)	**nack_period** (p. 2225)
	<<***extension***>> *(p. 116)* The period at which to send NACKs
rti::core::RtpsReliableReaderProtocol (p. 2221)	**receive_window_size** (p. 2226)
	<<***extension***>> *(p. 116)* The number of received out-of-order samples a reader can keep at a time
rti::core::RtpsReliableReaderProtocol (p. 2221)	**round_trip_time** (p. 2226)
	<<***extension***>> *(p. 116)* The duration from sending a NACK to receiving a repair of a sample
rti::core::RtpsReliableReaderProtocol (p. 2221)	**app_ack_period** (p. 2227)
	<<***extension***>> *(p. 116)* The period at which application-level acknowledgment messages are sent
rti::core::RtpsReliableReaderProtocol (p. 2221)	**min_app_ack_response_keep_duration** (p. 2227)
	<<***extension***>> *(p. 116)* Minimum duration for which application-level acknowledgment response data is kept
rti::core::RtpsReliableReaderProtocol (p. 2221)	**samples_per_app_ack** (p. 2228)
	<<***extension***>> *(p. 116)* The minimum number of samples acknowledged by one application-level acknowledgment message
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**low_watermark** (p. 2232)
	<<***extension***>> *(p. 116)* When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385) is considered to have changed.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**high_watermark** (p. 2233)
	<<***extension***>> *(p. 116)* When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, the **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385) is considered to have changed.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**virtual_heartbeat_period** (p. 2237)

	<<***extension***>> *(*p. *116)* The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**samples_per_virtual_heartbeat** (p. 2238)
	<<***extension***>> *(*p. *116)* The number of samples that a reliable writer has to publish before sending a virtual heartbeat.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_heartbeat_retries** (p. 2239)
	<<***extension***>> *(*p. *116)* The maximum number of *periodic* heartbeat retries before marking a remote reader as inactive.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**inactivate_nonprogressing_readers** (p. 2239)
	<<***extension***>> *(*p. *116)* Whether to treat remote readers as inactive when their NACKs do not progress
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**heartbeats_per_max_samples** (p. 2240)
	<<***extension***>> *(*p. *116)* The number of piggyback heartbeats sent per max send window.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**min_nack_response_delay** (p. 2242)
	<<***extension***>> *(*p. *116)* The minimum delay to respond to a NACK or NACK_FRAG.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_nack_response_delay** (p. 2242)
	<<***extension***>> *(*p. *116)* The maximum delay to respond to a NACK or NACK_FRAG.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**nack_suppression_duration** (p. 2243)
	<<***extension***>> *(*p. *116)* The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_bytes_per_nack_response** (p. 2244)
	<<***extension***>> *(*p. *116)* The maximum total message size when resending rejected samples or data fragments.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_enable_adaptive_sample_↩ keep_duration** (p. 2247)

	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Enables dynamic adjustment of sample keep duration in response to congestion.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_min_sample_keep_duration** (p. 2245)
	<<***extension***>> *(p. 116)* The minimum duration a sample is queued for ACK-disabled readers.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_max_sample_keep_duration** (p. 2246)
	<<***extension***>> *(p. 116)* The maximum duration a sample is queued for ACK-disabled readers.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_decrease_sample_keep_duration_factor** (p. 2248)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Controls rate of contraction of dynamic sample keep duration.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_increase_sample_keep_duration_factor** (p. 2248)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Controls rate of growth of dynamic sample keep duration.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**send_window_update_period** (p. 2251)
	<<***extension***>> *(p. 116)* Period in which send window may be dynamically changed
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**send_window_increase_factor** (p. 2252)
	<<***extension***>> *(p. 116)* Increases send window size by this percentage when reacting dynamically to network conditions
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**send_window_decrease_factor** (p. 2253)
	<<***extension***>> *(p. 116)* Decreases send window size by this percentage when reacting dynamically to network conditions
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**enable_multicast_periodic_heartbeat** (p. 2254)
	<<***extension***>> *(p. 116)* Whether periodic heartbeat messages are sent over multicast
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**multicast_resend_threshold** (p. 2254)

	<<***extension***>> *(*p. *116)* The minimum number of requesting readers needed to trigger a multicast resend
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_repair_piggyback_heartbeat** (p. 2255)
	<<***extension***>> *(*p. *116)* Prevents piggyback heartbeats from being sent with repair samples

6.68 Sample Memory Management QoS

Use these QoS fields to configure how Connext manages memory related to the samples that are stored and managed by the DataReader and DataWriter queues.

Use these QoS fields to configure how Connext manages memory related to the samples that are stored and managed by the DataReader and DataWriter queues.

Basic Settings

dds::core::policy::ResourceLimits (p. 2203)	**max_samples** (p. 2205)
	<<***basic***>> *(*p. *104)* <<***spec***>> *(*p. *112)* Represents the maximum samples the middleware can store for any one **dds::pub::↩ DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)).
dds::core::policy::ResourceLimits (p. 2203)	**max_instances** (p. 2206)
	<<***basic***>> *(*p. *104)* <<***spec***>> *(*p. *112)* Represents the maximum number of instances a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::ResourceLimits (p. 2203)	**initial_samples** (p. 2207)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Represents the initial samples the middleware will store for any one **dds::pub::↩ DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)).
dds::core::policy::ResourceLimits (p. 2203)	**initial_instances** (p. 2208)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Represents the initial number of instances a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) will manage.

Advanced Settings

rti::core::PersistentStorageSettings (p. 1927)	**writer_instance_cache_allocation** (p. 1934)
	<<***extension***>> *(*p. *116)* Configures the resource limits associated with the instance durable writer history cache.

rti::core::PersistentStorageSettings (p. 1927)	**writer_sample_cache_allocation** (p. 1935)
	<<***extension***>> *(p. 116)* Configures the resource limits associated with the sample durable writer history cache.
dds::core::policy::DurabilityService (p. 1443)	**history_kind** (p. 1445)
	<<***spec***>> *(p. 112)* The kind of history to apply in recouping durable data.
dds::core::policy::DurabilityService (p. 1443)	**history_depth** (p. 1446)
	<<***spec***>> *(p. 112)* Setting to use for the **dds::core::policy::Durability::writer_depth** (p. 1439) when recouping durable data.
dds::core::policy::DurabilityService (p. 1443)	**max_samples** (p. 1446)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_instances** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_samples_per_instance** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::ResourceLimits (p. 2203)	**max_samples_per_instance** (p. 2207)
	<<***spec***>> *(p. 112)* Represents the maximum number of samples of any one instance a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::WriterDataLifecycle (p. 2660)	**autodispose_unregistered_instances** (p. 2661)
	<<***spec***>> *(p. 112)* Boolean flag that controls whether or not a writer will also dispose of an instance when the **dds::pub::DataWriter** (p. 1138) unregisters from the instance by means of the unregister operations.
dds::core::policy::WriterDataLifecycle (p. 2660)	**autopurge_unregistered_instances_delay** (p. 2662)
	<<***extension***>> *(p. 116)* Maximum duration for which the **dds::pub::DataWriter** (p. 1138) will maintain information regarding an instance once it has unregistered the instance.
dds::core::policy::WriterDataLifecycle (p. 2660)	**autopurge_disposed_instances_delay** (p. 2664)
	<<***extension***>> *(p. 116)* Maximum duration for which the **dds::pub::DataWriter** (p. 1138) will maintain information regarding an instance once it has disposed the instance.

dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_nowriter_samples_delay** (p. 2140)
	<<***spec***>> *(p. 112)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain samples for an instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_↩ no_writers()** (p. 1617).
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_disposed_samples_delay** (p. 2141)
	<<***spec***>> *(p. 112)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain samples for an instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_↩ disposed()** (p. 1617).
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_disposed_instances_delay** (p. 2142)
	<<***extension***>> *(p. 116)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain information about a received instance once its `instance↩ _state` becomes **dds::sub::status::InstanceState↩ ::not_alive_disposed()** (p. 1617) and there are no samples for the instance in the DataReader queue.
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_nowriter_instances_delay** (p. 2143)
	<<***extension***>> *(p. 116)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain information about a received instance once its `instance↩ _state` becomes **dds::sub::status::InstanceState↩ ::not_alive_no_writers()** (p. 1617) and there are no samples for the instance in the DataReader queue.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragmented_samples** (p. 1092)
	<<***extension***>> *(p. 116)* The maximum number of samples for which the **dds::sub::DataReader** (p. 976) may store fragments at a given point in time.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_fragmented_samples** (p. 1093)
	<<***extension***>> *(p. 116)* The initial number of samples for which a **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragmented_samples_per_remote_writer** (p. 1094)
	<<***extension***>> *(p. 116)* The maximum number of samples per remote writer for which a **dds::sub::Data↩ Reader** (p. 976) may store fragments.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragments_per_sample** (p. 1094)

	<<***extension***>> *(p. 116)* Maximum number of fragments for a single sample.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**dynamically_allocate_fragmented_samples** (p. 1095)
	<<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_batches** (p. 1239)
	<<***extension***>> *(p. 116)* Represents the initial number of batches a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_batches** (p. 1240)
	<<***extension***>> *(p. 116)* Represents the maximum number of batches a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**writer_loaned_sample_allocation** (p. 1247)
	<<***extension***>> *(p. 116)* Represents the allocation settings of loaned samples managed by a **dds::pub↩ ::DataWriter** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initialize_writer_loaned_sample** (p. 1248)
	<<***extension***>> *(p. 116)* Whether or not to initialize loaned samples returned by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_reader_resource_limits** (p. 1285)
	<<***extension***>> *(p. 116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_reader_resource_limits** (p. 1287)
	<<***extension***>> *(p. 116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_reader_resource_limits** (p. 1289)
	<<***extension***>> *(p. 116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_reader_resource_limits** (p. 1313)
	<<***extension***>> *(p. 116)* Resource limits for the built-in topic participant configuration reader.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_samples** (p. 909)
	<<***extension***>> *(p. 116)* Initial number of samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_samples** (p. 909)

	<<***extension***>> *(*p. *116)* Maximum number of samples.

6.69 Sample Ordering

Use these QoS fields to configure how samples are ordered when stored and accessed by a DataReader.

Use these QoS fields to configure how samples are ordered when stored and accessed by a DataReader.

Basic Settings

dds::core::policy::DestinationOrder (p. 1258)	**kind** (p. 1261)
	<<***basic***>> *(*p. *104)* <<***spec***>> *(*p. *112)* Specifies the desired kind of destination order.

Advanced Settings

dds::core::policy::Presentation (p. 1942)	**access_scope** (p. 1947)
	<<***spec***>> *(*p. *112)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
dds::core::policy::Presentation (p. 1942)	**coherent_access** (p. 1947)
	<<***spec***>> *(*p. *112)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.
dds::core::policy::Presentation (p. 1942)	**ordered_access** (p. 1948)
	<<***spec***>> *(*p. *112)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
dds::core::policy::Presentation (p. 1942)	**drop_incomplete_coherent_set** (p. 1949)
	<<***extension***>> *(*p. *116)* Indicates whether or not a **dds::sub::DataReader** (p. 976) should drop samples from an incomplete coherent set (one for which not all the samples were received). Such samples are reported as lost in the SAMPLE_LOST Status.
dds::core::policy::DestinationOrder (p. 1258)	**scope** (p. 1262)
	<<***spec***>> *(*p. *112)* Specifies the desired scope of the source destination order.
dds::core::policy::DestinationOrder (p. 1258)	**source_timestamp_tolerance** (p. 1262)

	<<***extension***>> *(p. 116)* Allowed tolerance between source timestamps of consecutive samples.

6.70 Security

Use these QoS fields to enable and configure some security-related capabilities in RTI Connext.

Use these QoS fields to enable and configure some security-related capabilities in RTI Connext.

For a complete security solution, you should use the Security Plugins. See the `RTI Security Plugins User's Manual` for more information.

The QoS listed on this page mostly relate to features that are enabled when the Security Plugins are being used.

Basic Settings

rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**security_event_forwarding_level** (p. 1734)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Log messages with **rti::config::LogFacility::security_event** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

Advanced Settings

rti::core::policy::TypeSupport (p. 2574)	**cdr_padding_kind** (p. 2576)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Determines whether or not the padding bytes will be set to zero during CDR serialization.
rti::core::policy::WireProtocol (p. 2630)	**compute_crc** (p. 2640)
	<<***extension***>> *(p. 116)* Adds RTPS CRC submessage to every message when this field is set to true.
rti::core::policy::WireProtocol (p. 2630)	**check_crc** (p. 2640)
	<<***extension***>> *(p. 116)* Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to true.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_writer** (p. 1307)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.

rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_writer_publish_mode** (p. 1309)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in secure volatile writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_reader** (p. 1309)
	<<***extension***>> *(p. 116)* RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
dds::core::policy::DataTag (p. 1132)	**tags** (p. 1137)
	<<***spec***>> *(p. 112)* Sequence of data tags.

6.71 Throughput

Use these QoS fields to configure features which may be helpful to increase throughput.

Use these QoS fields to configure features which may be helpful to increase throughput.

Basic Settings

rti::core::policy::Batch (p. 880)	**enable** (p. 882)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Specifies whether or not batching is enabled.
rti::core::policy::Batch (p. 880)	**max_data_bytes** (p. 882)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The maximum cumulative length of all serialized samples in a batch.
rti::core::policy::Batch (p. 880)	**max_samples** (p. 883)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The maximum number of samples in a batch.

Advanced Settings

rti::core::policy::Batch (p. 880)	**max_flush_delay** (p. 884)
	<<***extension***>> *(p. 116)* The maximum flush delay.
rti::core::policy::Batch (p. 880)	**source_timestamp_resolution** (p. 885)
	<<***extension***>> *(p. 116)* Batch source timestamp resolution.
rti::core::policy::Batch (p. 880)	**thread_safe_write** (p. 886)
	<<***extension***>> *(p. 116)* Determines whether or not the write operation is thread safe.

6.72 Transports

Use these QoS fields to configure aspects of the transports (udp, shared memory, etc.) used for communication.

Use these QoS fields to configure aspects of the transports (udp, shared memory, etc.) used for communication.

Basic Settings

rti::core::policy::TransportBuiltin (p. 2533)	**mask** (p. 2536)
	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* Specifies the built-in transports that are registered automatically when the **dds::domain::DomainParticipant** (p. 1326) is enabled.
rti::core::policy::TransportMulticast (p. 2543)	**value** (p. 2545)
	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* A sequence of multicast communications settings.
rti::core::policy::Discovery (p. 1265)	**initial_peers** (p. 1268)
	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.
rti::core::policy::Discovery (p. 1265)	**multicast_receive_addresses** (p. 1268)
	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* Specifies the multicast group addresses on which discovery-related **metatraffic** can be received by the DomainParticipant.
rti::core::policy::Discovery (p. 1265)	**accept_unknown_peers** (p. 1270)
	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* Whether to accept a new participant that is not in the initial peers list.

Advanced Settings

dds::core::policy::TransportPriority (p. 2552)	**value** (p. 2553)
	*<<**spec**>>* (p. *112)* This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.
rti::core::policy::TransportSelection (p. 2554)	**enabled_transports** (p. 2555)
	*<<**extension**>>* (p. *116)* A sequence of transport aliases that specifies the transport instances available for use by the entity.
rti::core::policy::TransportUnicast (p. 2556)	**value** (p. 2558)
	*<<**extension**>>* (p. *116)* A sequence of unicast communication settings.

rti::core::policy::TransportMulticastMapping (p. 2547)	**value** (p. 2549)
	<<***extension***>> *(p. 116)* A sequence of multicast communication mappings.
rti::core::policy::TransportMulticast (p. 2543)	**kind** (p. 2546)
	<<***extension***>> *(p. 116)* A value that specifies a way to determine how to obtain the multicast address.
rti::core::policy::Discovery (p. 1265)	**enabled_transports** (p. 1267)
	<<***extension***>> *(p. 116)* The transports available for use by the Discovery mechanism.
rti::core::policy::Discovery (p. 1265)	**metatraffic_transport_priority** (p. 1269)
	<<***extension***>> *(p. 116)* The transport priority to use for the Discovery meta-traffic.
rti::core::policy::Discovery (p. 1265)	**enable_endpoint_discovery** (p. 1271)
	[DEPRECATED] (p. **296)** <<***extension***>> *(p. 116)* Whether to automatically enable endpoint discovery for all the remote participants.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**transport_info_list_max_length** (p. 1428)
	<<***extension***>> *(p. 116)* Maximum number of installed transports to send and receive information about in **dds::topic::ParticipantBuiltinTopic↩ Data::transport_info** (p. 1914).
rti::core::policy::DiscoveryConfig (p. 1271)	**locator_reachability_assert_period** (p. 1305)
	<<***extension***>> *(p. 116)* Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.
rti::core::policy::DiscoveryConfig (p. 1271)	**locator_reachability_change_detection_period** (p. 1306)
	<<***extension***>> *(p. 116)* Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.
rti::core::policy::DiscoveryConfig (p. 1271)	**dns_tracker_polling_period** (p. 1311)
	<<***extension***>> *(p. 116)* Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**enable_multicast_periodic_heartbeat** (p. 2254)
	<<***extension***>> *(p. 116)* Whether periodic heartbeat messages are sent over multicast

rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**multicast_resend_threshold** (p. 2254)
	<<***extension***>> *(p. 116)* The minimum number of requesting readers needed to trigger a multicast resend
rti::core::ChannelSettings (p. 921)	**multicast_settings** (p. 922)
	<<***extension***>> *(p. 116)* A sequence of **rti::core::TransportMulticastSettings** (p. 2549) used to configure the multicast addresses associated with a channel.
rti::core::ChannelSettings (p. 921)	**priority** (p. 924)
	<<***extension***>> *(p. 116)* Publication priority

6.73 Topic Queries

Use these QoS fields to configure the topic query feature.

Use these QoS fields to configure the topic query feature.

See `Querying Data`, in the *Core Libraries User's Manual*, for more information about this feature.

Basic Settings

rti::core::policy::TopicQueryDispatch (p. 2522)	**publication_period** (p. 2525)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Sets the periodic interval at which samples are published.

Advanced Settings

rti::core::policy::AsynchronousPublisher (p. 832)	**disable_topic_query_publication** (p. 838)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Disable topic query publication.
rti::core::policy::AsynchronousPublisher (p. 832)	**topic_query_publication_thread** (p. 838)
	<<***extension***>> *(p. 116)* Settings of the **rti::sub::TopicQuery** (p. 2516) publication thread.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_topic_queries** (p. 1102)
	<<***extension***>> *(p. 116)* The initial number of TopicQueries allocated by a **dds::sub::DataReader** (p. 976).

rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_topic_queries** (p. 1103)
	<<***extension***>> *(p. 116)* The maximum number of active TopicQueries that a **dds::sub::DataReader** (p. 976) can create.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_active_topic_queries** (p. 1246)
	<<***extension***>> *(p. 116)* Represents the initial number of active topic queries a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_active_topic_queries** (p. 1246)
	<<***extension***>> *(p. 116)* Represents the maximum number of active topic queries a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_topic_query_allocation** (p. 1429)
	<<***extension***>> *(p. 116)* Allocation settings applied to remote TopicQueries.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_topic_query_hash_buckets** (p. 1430)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Number of hash buckets for remote TopicQueries.
rti::core::policy::TopicQueryDispatch (p. 2522)	**enable** (p. 2524)
	<<***extension***>> *(p. 116)* Allows this writer to dispatch TopicQueries
rti::core::policy::TopicQueryDispatch (p. 2522)	**samples_per_period** (p. 2525)
	<<***extension***>> *(p. 116)* Sets the maximum number of samples to publish in each publication_period.

6.74 Type System

Use these QoS fields to configure aspects of the type system.

Use these QoS fields to configure aspects of the type system.

These settings include how types are sent during discovery, what criteria is used when types should be matched, and how data should be serialized.

Advanced Settings

dds::core::policy::DataRepresentation (p. 1113)	**value** (p. 1116)
	<<***spec***>> *(p. 112)* Sequence of representation identifiers.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**kind** (p. 2565)
	<<***spec***>> *(p. 112)* Type consistency kind.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_sequence_bounds** (p. 2566)
	<<***spec***>> *(p. 112)* Controls whether sequence bounds are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_string_bounds** (p. 2567)
	<<***spec***>> *(p. 112)* Controls whether string bounds are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_member_names** (p. 2567)
	<<***spec***>> *(p. 112)* Controls whether member names are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**prevent_type_widening** (p. 2568)
	<<***spec***>> *(p. 112)* Controls whether type widening is allowed.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**force_type_validation** (p. 2568)
	<<***spec***>> *(p. 112)* Controls whether type information must be available in order to complete matching between a **dds::pub::DataWriter** (p. 1138) and a **dds::sub::DataReader** (p. 976).
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_enum_literal_names** (p. 2569)
	<<***spec***>> *(p. 112)* Controls whether enumeration constant names are taken into consideration for type assignability.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_code_max_serialized_length** (p. 1418)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Maximum size of serialized string for type code.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_object_max_serialized_length** (p. 1419)

	<<***extension***>> *(p. 116)* The maximum length, in bytes, that the buffer to serialize a TypeObject v1 can consume.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_object_max_deserialized_length** (p. 1420)
	<<***extension***>> *(p. 116)* The maximum number of bytes that a deserialized TypeObject v1 can consume.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**deserialized_type_object_dynamic_allocation_threshold** (p. 1420)
	<<***extension***>> *(p. 116)* A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject v1.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**serialized_type_object_dynamic_allocation_threshold** (p. 1421)
	<<***extension***>> *(p. 116)* A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject v1.
rti::core::policy::DiscoveryConfig (p. 1271)	**endpoint_type_object_lb_serialization_threshold** (p. 1310)
	<<***extension***>> *(p. 116)* Controls when/if the TypeObject v1 is compressed when propagated in Simple Endpoint Discovery.

6.75 QoS Mutability

Some QoS are mutable after an entity is enabled, and others are not.

Modules

- **Immutable QoS**

  This page lists all QoS that are immutable after an entity has been enabled.
- **Mutable QoS**

  This page lists all QoS that are always mutable.

6.75.1 Detailed Description

Some QoS are mutable after an entity is enabled, and others are not.

6.76 Immutable QoS

This page lists all QoS that are immutable after an entity has been enabled.

This page lists all QoS that are immutable after an entity has been enabled.

QoS fall into one of three mutability categories:

- Always mutable, wheter an entity is enabled or not (see **Mutable QoS** (p. 280)).
- Immutable after entity creation.
- *Mutable* after creation but *immutable* after an entity is enabled.

The QoS that fall into the second two categories are listed on this page.

See also

Changeable (p. 514)

Basic Settings

dds::core::policy::Durability (p. 1433)	**kind** (p. 1437)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of durability.
dds::core::policy::Durability (p. 1433)	**writer_depth** (p. 1439)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Indicates the number of samples per instance that a durable **dds::pub::DataWriter** (p. 1138) will send to a late-joining **dds::sub::DataReader** (p. 976).
dds::core::policy::DataRepresentation (p. 1113)	**compression_settings** (p. 1117)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Structure that contains the compression settings.
dds::core::policy::Ownership (p. 1899)	**kind** (p. 1904)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of ownership
dds::core::policy::Liveliness (p. 1645)	**kind** (p. 1649)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of liveliness desired.
dds::core::policy::Liveliness (p. 1645)	**lease_duration** (p. 1649)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The duration within which a **dds::pub::DataWriter** (p. 1138) must be asserted, or else it is assumed to be not alive.

dds::core::policy::Liveliness (p. 1645)	**assertions_per_lease_duration** (p. 1651)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The number of assertions a **dds::pub::DataWriter** (p. 1138) will send during a its **dds::core::policy::Liveliness**↩ **::lease_duration** (p. 1649).
dds::core::policy::Reliability (p. 2149)	**kind** (p. 2153)
	<<***basic***>> (p. *104)* <<***spec***>> (p. *112)* Kind of reliability.
dds::core::policy::Reliability (p. 2149)	**max_blocking_time** (p. 2153)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* The maximum time a DataWriter can block on a **write()** (p. 1179) call.
dds::core::policy::DestinationOrder (p. 1258)	**kind** (p. 1261)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Specifies the desired kind of destination order.
dds::core::policy::History (p. 1602)	**kind** (p. 1604)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Specifies the kind of history to be kept.
dds::core::policy::History (p. 1602)	**depth** (p. 1605)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Specifies the number of samples per instance to be kept, when the `kind` is **dds::core::policy::HistoryKind_def::KEEP**↩ **_LAST** (p. 1607)
dds::core::policy::ResourceLimits (p. 2203)	**max_samples** (p. 2205)
	<<***basic***>> (p. *104)* <<***spec***>> *(p. 112)* Represents the maximum samples the middleware can store for any one **dds::pub::DataWriter** (p. 1138) (or **dds::sub::**↩ **DataReader** (p. 976)).
dds::core::policy::ResourceLimits (p. 2203)	**max_instances** (p. 2206)
	<<***basic***>> *(p. 104)* <<***spec***>> (p. *112)* Represents the maximum number of instances a **dds::pub::Data**↩ **Writer** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::ResourceLimits (p. 2203)	**initial_samples** (p. 2207)
	<<***basic***>> (p. *104)* <<***extension***>> *(p. 116)* Represents the initial samples the middleware will store for any one **dds::pub::DataWriter** (p. 1138) (or **dds::sub**↩ **::DataReader** (p. 976)).
dds::core::policy::ResourceLimits (p. 2203)	**initial_instances** (p. 2208)

	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Represents the initial number of instances a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) will manage.
rti::core::policy::PublishMode (p. 2013)	**kind** (p. 2016)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Publishing mode
rti::core::policy::WireProtocol (p. 2630)	**participant_id** (p. 2635)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* A value used to distinguish among different participants belonging to the same domain on the same host.
rti::core::policy::TransportBuiltin (p. 2533)	**mask** (p. 2536)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Specifies the built-in transports that are registered automatically when the **dds::domain::DomainParticipant** (p. 1326) is enabled.
rti::core::policy::TransportMulticast (p. 2543)	**value** (p. 2545)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* A sequence of multicast communications settings.
rti::core::policy::Discovery (p. 1265)	**initial_peers** (p. 1268)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.
rti::core::policy::Discovery (p. 1265)	**multicast_receive_addresses** (p. 1268)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the DomainParticipant.
rti::core::policy::Discovery (p. 1265)	**accept_unknown_peers** (p. 1270)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Whether to accept a new participant that is not in the initial peers list.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_liveliness_lease_duration** (p. 1279)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The liveliness lease duration for the participant.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_liveliness_assert_period** (p. 1279)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The period to assert liveliness for the participant.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_announcement_period** (p. 1280)

	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
rti::core::policy::DiscoveryConfig (p. 1271)	**initial_participant_announcements** (p. 1282)
	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* The number of initial announcements sent when a participant is first enabled.
rti::core::policy::DiscoveryConfig (p. 1271)	**new_remote_participant_announcements** (p. 1283)
	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* The number of participant announcements sent when a remote participant is newly discovered.
rti::core::policy::DiscoveryConfig (p. 1271)	**min_initial_participant_announcement_period** (p. 1283)
	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
rti::core::policy::DiscoveryConfig (p. 1271)	**max_initial_participant_announcement_period** (p. 1284)
	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**heartbeat_period** (p. 2234)
	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* The period at which to send heartbeats.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**fast_heartbeat_period** (p. 2235)
	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**late_joiner_heartbeat_period** (p. 2236)
	*<<**basic**>>* (p. *104)* *<<**extension**>>* (p. *116)* An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**min_send_window_size** (p. 2249)

	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Minimum size of send window of unacknowledged samples
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_send_window_size** (p. 2250)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Maximum size of send window of unacknowledged samples
rti::core::policy::EntityName (p. 1530)	**name** (p. 1531)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The name of the entity.
rti::core::policy::Batch (p. 880)	**enable** (p. 882)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Specifies whether or not batching is enabled.
rti::core::policy::Batch (p. 880)	**max_data_bytes** (p. 882)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The maximum cumulative length of all serialized samples in a batch.
rti::core::policy::Batch (p. 880)	**max_samples** (p. 883)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* The maximum number of samples in a batch.
rti::core::QosProviderParams (p. 2049)	**string_profile** (p. 2050)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sequence of strings containing a XML document to load.
rti::core::QosProviderParams (p. 2049)	**url_profile** (p. 2050)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sequence of **URL groups** (p. 304) containing a set of XML documents to load.
rti::core::QosProviderParams (p. 2049)	**ignore_user_profile** (p. 2051)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Ignores the file USER_QOS_PROFILES.xml in the current working directory.
rti::core::QosProviderParams (p. 2049)	**ignore_environment_profile** (p. 2052)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Ignores the value of the **NDDS_QOS_PROFILES** (p. 304).
rti::core::QosProviderParams (p. 2049)	**ignore_resource_profile** (p. 2052)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Ignores the file NDDS_QOS_PROFILES.xml.
rti::core::policy::TopicQueryDispatch (p. 2522)	**publication_period** (p. 2525)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Sets the periodic interval at which samples are published.

Advanced Settings

rti::core::policy::Availability (p. 868)	**enable_required_subscriptions** (p. 871)
	<<***extension***>> *(p. 116)* Enables support for required subscriptions in a **dds::pub::DataWriter** (p. 1138)
dds::core::policy::Durability (p. 1433)	**direct_communication** (p. 1438)
	<<***extension***>> *(p. 116)* Indicates whether or not a TRANSIENT or PERSISTENT **dds::sub::DataReader** (p. 976) should receive samples directly from a TRANSIENT or PERSISTENT **dds::pub::DataWriter** (p. 1138)
dds::core::policy::Durability (p. 1433)	**storage_settings** (p. 1440)
	<<***extension***>> *(p. 116)* Configures durable writer history and durable reader state.
rti::core::PersistentStorageSettings (p. 1927)	**enable** (p. 1929)
	<<***extension***>> *(p. 116)* Enables durable writer history in a **dds::pub::DataWriter** (p. 1138) and durable reader state in a **dds::sub::DataReader** (p. 976).
rti::core::PersistentStorageSettings (p. 1927)	**file_name** (p. 1929)
	<<***extension***>> *(p. 116)* The file name where the durable writer history or durable reader state will be stored.
rti::core::PersistentStorageSettings (p. 1927)	**trace_file_name** (p. 1931)
	<<***extension***>> *(p. 116)* The file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.
rti::core::PersistentStorageSettings (p. 1927)	**journal_kind** (p. 1932)
	<<***extension***>> *(p. 116)* Sets the journal mode of the persistent storage.
rti::core::PersistentStorageSettings (p. 1927)	**synchronization_kind** (p. 1933)
	<<***extension***>> *(p. 116)* Sets the level of synchronization with the physical disk.
rti::core::PersistentStorageSettings (p. 1927)	**vacuum** (p. 1933)
	<<***extension***>> *(p. 116)* Sets the auto-vacuum status of the storage.
rti::core::PersistentStorageSettings (p. 1927)	**restore** (p. 1934)
	<<***extension***>> *(p. 116)* Indicates if the persisted writer history or reader state must be restored.

rti::core::PersistentStorageSettings (p. 1927)	**writer_instance_cache_allocation** (p. 1934)
	<<***extension***>> *(p. 116)* Configures the resource limits associated with the instance durable writer history cache.
rti::core::PersistentStorageSettings (p. 1927)	**writer_sample_cache_allocation** (p. 1935)
	<<***extension***>> *(p. 116)* Configures the resource limits associated with the sample durable writer history cache.
rti::core::PersistentStorageSettings (p. 1927)	**writer_memory_state** (p. 1936)
	<<***extension***>> *(p. 116)* Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.
rti::core::PersistentStorageSettings (p. 1927)	**reader_checkpoint_frequency** (p. 1937)
	<<***extension***>> *(p. 116)* Controls how often the reader state is stored into the database.
dds::core::policy::DurabilityService (p. 1443)	**service_cleanup_delay** (p. 1445)
	<<***spec***>> *(p. 112)* Controls when the service is able to remove all information regarding a data instances.
dds::core::policy::DurabilityService (p. 1443)	**history_kind** (p. 1445)
	<<***spec***>> *(p. 112)* The kind of history to apply in recouping durable data.
dds::core::policy::DurabilityService (p. 1443)	**history_depth** (p. 1446)
	<<***spec***>> *(p. 112)* Setting to use for the **dds::core::policy::Durability::writer_depth** (p. 1439) when recouping durable data.
dds::core::policy::DurabilityService (p. 1443)	**max_samples** (p. 1446)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_instances** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_samples_per_instance** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
rti::core::CompressionSettings (p. 943)	**compression_ids** (p. 945)
	<<***extension***>> *(p. 116)* Mask that represents the compression algorithms enabled.
rti::core::CompressionSettings (p. 943)	**writer_compression_level** (p. 946)

	<<***extension***>> *(*p. *116)* The level of compression to use when compressing data.
rti::core::CompressionSettings (p. 943)	**writer_compression_threshold** (p. 947)
	<<***extension***>> *(*p. *116)* The threshold, in bytes, above which a serialized sample will be eligible to be compressed.
dds::core::policy::DataRepresentation (p. 1113)	**value** (p. 1116)
	<<***spec***>> *(*p. *112)* Sequence of representation identifiers.
dds::core::policy::Presentation (p. 1942)	**access_scope** (p. 1947)
	<<***spec***>> *(*p. *112)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
dds::core::policy::Presentation (p. 1942)	**coherent_access** (p. 1947)
	<<***spec***>> *(*p. *112)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.
dds::core::policy::Presentation (p. 1942)	**ordered_access** (p. 1948)
	<<***spec***>> *(*p. *112)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
dds::core::policy::Presentation (p. 1942)	**drop_incomplete_coherent_set** (p. 1949)
	<<***extension***>> *(*p. *116)* Indicates whether or not a **dds::sub::DataReader** (p. 976) should drop samples from an incomplete coherent set (one for which not all the samples were received). Such samples are reported as lost in the SAMPLE_LOST Status.
dds::core::policy::Reliability (p. 2149)	**acknowledgment_kind** (p. 2154)
	<<***extension***>> *(*p. *116)* Kind of reliable acknowledgment
dds::core::policy::Reliability (p. 2149)	**instance_state_consistency_kind** (p. 2155)
	<<***extension***>> *(*p. *116)* Whether instance state consistency is enabled
dds::core::policy::Reliability (p. 2149)	**exact_reliability_kind_match** (p. 2155)
	<<***extension***>> *(*p. *116)* Whether an exact reliability kind match is required for endpoint matching.
dds::core::policy::DestinationOrder (p. 1258)	**scope** (p. 1262)
	<<***spec***>> *(*p. *112)* Specifies the desired scope of the source destination order.

dds::core::policy::DestinationOrder (p. 1258)	**source_timestamp_tolerance** (p. 1262)
	<<***extension***>> *(p. 116)* Allowed tolerance between source timestamps of consecutive samples.
dds::core::policy::ResourceLimits (p. 2203)	**max_samples_per_instance** (p. 2207)
	<<***spec***>> *(p. 112)* Represents the maximum number of samples of any one instance a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage.
dds::core::policy::ResourceLimits (p. 2203)	**instance_hash_buckets** (p. 2208)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for instances.
dds::core::policy::TransportPriority (p. 2552)	**value** (p. 2553)
	<<***spec***>> *(p. 112)* This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.
rti::core::policy::TypeSupport (p. 2574)	**plugin_data** (p. 2575)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Value to pass into the type plugin's de-/serialization function
rti::core::policy::TypeSupport (p. 2574)	**cdr_padding_kind** (p. 2576)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Determines whether or not the padding bytes will be set to zero during CDR serialization.
rti::core::policy::PublishMode (p. 2013)	**flow_controller_name** (p. 2016)
	<<***extension***>> *(p. 116)* Name of the associated flow controller
rti::core::policy::PublishMode (p. 2013)	**priority** (p. 2017)
	<<***extension***>> *(p. 116)* Publication priority
rti::core::policy::AsynchronousPublisher (p. 832)	**disable_asynchronous_write** (p. 835)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Disable asynchronous publishing.
rti::core::policy::AsynchronousPublisher (p. 832)	**thread** (p. 835)
	<<***extension***>> *(p. 116)* Settings of the publishing thread.
rti::core::policy::AsynchronousPublisher (p. 832)	**disable_asynchronous_batch** (p. 836)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Disable asynchronous batch flushing.
rti::core::policy::AsynchronousPublisher (p. 832)	**asynchronous_batch_thread** (p. 837)

	<<***extension***>> *(*p. *116)* Settings of the batch flushing thread.
rti::core::policy::AsynchronousPublisher (p. 832)	**disable_topic_query_publication** (p. 838)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Disable topic query publication.
rti::core::policy::AsynchronousPublisher (p. 832)	**topic_query_publication_thread** (p. 838)
	<<***extension***>> *(*p. *116)* Settings of the **rti::sub::TopicQuery** (p. 2516) publication thread.
dds::core::policy::WriterDataLifecycle (p. 2660)	**autodispose_unregistered_instances** (p. 2661)
	<<***spec***>> *(*p. *112)* Boolean flag that controls whether or not a writer will also dispose of an instance when the **dds::pub::DataWriter** (p. 1138) unregisters from the instance by means of the unregister operations.
dds::core::policy::WriterDataLifecycle (p. 2660)	**autopurge_unregistered_instances_delay** (p. 2662)
	<<***extension***>> *(*p. *116)* Maximum duration for which the **dds::pub::DataWriter** (p. 1138) will maintain information regarding an instance once it has unregistered the instance.
dds::core::policy::WriterDataLifecycle (p. 2660)	**autopurge_disposed_instances_delay** (p. 2664)
	<<***extension***>> *(*p. *116)* Maximum duration for which the **dds::pub::DataWriter** (p. 1138) will maintain information regarding an instance once it has disposed the instance.
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_nowriter_samples_delay** (p. 2140)
	<<***spec***>> *(*p. *112)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain samples for an instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617).
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_disposed_samples_delay** (p. 2141)
	<<***spec***>> *(*p. *112)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain samples for an instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617).
dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_disposed_instances_delay** (p. 2142)
	<<***extension***>> *(*p. *116)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain information about a received instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) and there are no samples for the instance in the DataReader queue.

dds::core::policy::ReaderDataLifecycle (p. 2137)	**autopurge_nowriter_instances_delay** (p. 2143)
	<<***extension***>> *(p. 116)* Minimum duration for which the **dds::sub::DataReader** (p. 976) will maintain information about a received instance once its `instance_state` becomes **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) and there are no samples for the instance in the DataReader queue.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**kind** (p. 2565)
	<<***spec***>> *(p. 112)* Type consistency kind.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_sequence_bounds** (p. 2566)
	<<***spec***>> *(p. 112)* Controls whether sequence bounds are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_string_bounds** (p. 2567)
	<<***spec***>> *(p. 112)* Controls whether string bounds are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_member_names** (p. 2567)
	<<***spec***>> *(p. 112)* Controls whether member names are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**prevent_type_widening** (p. 2568)
	<<***spec***>> *(p. 112)* Controls whether type widening is allowed.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**force_type_validation** (p. 2568)
	<<***spec***>> *(p. 112)* Controls whether type information must be available in order to complete matching between a **dds::pub::DataWriter** (p. 1138) and a **dds::sub::DataReader** (p. 976).
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_enum_literal_names** (p. 2569)
	<<***spec***>> *(p. 112)* Controls whether enumeration constant names are taken into consideration for type assignability.
rti::core::policy::WireProtocol (p. 2630)	**rtps_host_id** (p. 2636)
	<<***extension***>> *(p. 116)* The RTPS Host ID of the domain participant.
rti::core::policy::WireProtocol (p. 2630)	**rtps_app_id** (p. 2636)

	<<***extension***>> *(*p. *116)* The RTPS App ID of the domain participant.
rti::core::policy::WireProtocol (p. 2630)	**rtps_instance_id** (p. 2637)
	<<***extension***>> *(*p. *116)* The RTPS Instance ID of the **dds::domain::DomainParticipant** (p. 1326).
rti::core::policy::WireProtocol (p. 2630)	**rtps_well_known_ports** (p. 2638)
	<<***extension***>> *(*p. *116)* Configures the RTPS well-known port mappings.
rti::core::policy::WireProtocol (p. 2630)	**rtps_reserved_port_mask** (p. 2639)
	<<***extension***>> *(*p. *116)* Specifies which well-known ports to reserve when enabling the participant.
rti::core::policy::WireProtocol (p. 2630)	**rtps_auto_id_kind** (p. 2639)
	<<***extension***>> *(*p. *116)* Kind of auto mechanism used to calculate the GUID prefix.
rti::core::policy::WireProtocol (p. 2630)	**compute_crc** (p. 2640)
	<<***extension***>> *(*p. *116)* Adds RTPS CRC submessage to every message when this field is set to true.
rti::core::policy::WireProtocol (p. 2630)	**check_crc** (p. 2640)
	<<***extension***>> *(*p. *116)* Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to true.
rti::core::policy::TransportSelection (p. 2554)	**enabled_transports** (p. 2555)
	<<***extension***>> *(*p. *116)* A sequence of transport aliases that specifies the transport instances available for use by the entity.
rti::core::policy::TransportUnicast (p. 2556)	**value** (p. 2558)
	<<***extension***>> *(*p. *116)* A sequence of unicast communication settings.
rti::core::policy::TransportMulticastMapping (p. 2547)	**value** (p. 2549)
	<<***extension***>> *(*p. *116)* A sequence of multicast communication mappings.
rti::core::policy::TransportMulticast (p. 2543)	**kind** (p. 2546)
	<<***extension***>> *(*p. *116)* A value that specifies a way to determine how to obtain the multicast address.
rti::core::policy::Discovery (p. 1265)	**enabled_transports** (p. 1267)
	<<***extension***>> *(*p. *116)* The transports available for use by the Discovery mechanism.

rti::core::policy::Discovery (p. 1265)	**metatraffic_transport_priority** (p. 1269)
	<<***extension***>> *(p. 116)* The transport priority to use for the Discovery meta-traffic.
rti::core::policy::Discovery (p. 1265)	**enable_endpoint_discovery** (p. 1271)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Whether to automatically enable endpoint discovery for all the remote participants.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_writers** (p. 1085)
	<<***extension***>> *(p. 116)* The maximum number of remote writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_writers_per_instance** (p. 1086)
	<<***extension***>> *(p. 116)* The maximum number of remote writers from which a **dds::sub::DataReader** (p. 976) may read a single instance.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_samples_per_remote_writer** (p. 1087)
	<<***extension***>> *(p. 116)* The maximum number of out-of-order samples from a given remote **dds::pub::DataWriter** (p. 1138) that a **dds::sub::DataReader** (p. 976) may store when maintaining a reliable connection to the **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_infos** (p. 1087)
	<<***extension***>> *(p. 116)* The maximum number of info units that a **dds::sub::DataReader** (p. 976) can use to store **dds::sub::SampleInfo** (p. 2285).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_remote_writers** (p. 1088)
	<<***extension***>> *(p. 116)* The initial number of remote writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_remote_writers_per_instance** (p. 1089)
	<<***extension***>> *(p. 116)* The initial number of remote writers from which a **dds::sub::DataReader** (p. 976) may read a single instance.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_infos** (p. 1089)

	<<***extension***>> *(p. 116)* The initial number of info units that a **dds::sub::DataReader** (p. 976) can have, which are used to store **dds::sub::SampleInfo** (p. 2285).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_outstanding_reads** (p. 1090)
	<<***extension***>> *(p. 116)* The initial number of outstanding calls to read/take (or one of their variants) on the same **dds::sub::DataReader** (p. 976) for which memory has not been returned by calling **dds::sub::LoanedSamples::return_loan()** (p. 1667).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_outstanding_reads** (p. 1091)
	<<***extension***>> *(p. 116)* The maximum number of outstanding read/take calls (or one of their variants) on the same **dds::sub::DataReader** (p. 976) for which memory has not been returned by calling **dds::sub::LoanedSamples::return_loan()** (p. 1667).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_samples_per_read** (p. 1091)
	<<***extension***>> *(p. 116)* The maximum number of data samples that the application can receive from the middleware in a single call to **dds::sub::DataReader::read** (p. 996) or **dds::sub::DataReader::take** (p. 997). If more data exists in the middleware, the application will need to issue multiple read/take calls.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**disable_fragmentation_support** (p. 1092)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) can receive fragmented samples.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragmented_samples** (p. 1092)
	<<***extension***>> *(p. 116)* The maximum number of samples for which the **dds::sub::DataReader** (p. 976) may store fragments at a given point in time.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_fragmented_samples** (p. 1093)
	<<***extension***>> *(p. 116)* The initial number of samples for which a **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragmented_samples_per_remote_writer** (p. 1094)

	<<***extension***>> *(p. 116)* The maximum number of samples per remote writer for which a **dds::sub::Data↩ Reader** (p. 976) may store fragments.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_fragments_per_sample** (p. 1094)
	<<***extension***>> *(p. 116)* Maximum number of fragments for a single sample.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**dynamically_allocate_fragmented_samples** (p. 1095)
	<<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_total_instances** (p. 1096)
	<<***extension***>> *(p. 116)* Maximum number of instances for which a DataReader will keep state.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_virtual_writers** (p. 1097)
	<<***extension***>> *(p. 116)* The maximum number of remote virtual writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_remote_virtual_writers** (p. 1097)
	<<***extension***>> *(p. 116)* The initial number of remote virtual writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_virtual_writers_per_instance** (p. 1098)
	<<***extension***>> *(p. 116)* The maximum number of virtual remote writers that can be associated with an instance.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_remote_virtual_writers_per_instance** (p. 1099)
	<<***extension***>> *(p. 116)* The initial number of virtual remote writers per instance.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_remote_writers_per_sample** (p. 1099)
	<<***extension***>> *(p. 116)* The maximum number of remote writers allowed to write the same sample.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_query_condition_filters** (p. 1100)

	<<***extension***>> *(*p. *116)* The maximum number of query condition filters a reader is allowed.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_app_ack_response_length** (p. 1101)
	<<***extension***>> *(*p. *116)* Maximum length of application-level acknowledgment response data
rti::core::policy::DataReaderResourceLimits (p. 1081)	**keep_minimum_state_for_instances** (p. 1101)
	<<***extension***>> *(*p. *116)* Whether or not keep a minimum instance state for up to **rti::core::policy::↩ DataReaderResourceLimits::max_total_instances** (p. 1096).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**initial_topic_queries** (p. 1102)
	<<***extension***>> *(*p. *116)* The initial number of Topic↩ Queries allocated by a **dds::sub::DataReader** (p. 976).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**max_topic_queries** (p. 1103)
	<<***extension***>> *(*p. *116)* The maximum number of active TopicQueries that a **dds::sub::DataReader** (p. 976) can create.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**shmem_ref_transfer_mode_attached_segment_↩ allocation** (p. 1106)
	<<***extension***>> *(*p. *116)* Allocation resource for the shared memory segments attached by the **dds::sub↩ ::DataReader** (p. 976).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**instance_replacement** (p. 1106)
	<<***extension***>> *(*p. *116)* Sets the kind of instances allowed to be replaced for each instance state (**dds↩ ::sub::status::InstanceState** (p. 1615)) when a Data↩ Reader reaches **dds::core::policy::ResourceLimits↩ ::max_instances** (p. 2206).
rti::core::policy::DataReaderResourceLimits (p. 1081)	**autopurge_remote_not_alive_writer_delay** (p. 1103)
	<<***extension***>> *(*p. *116)* Maximum duration for which the **dds::sub::DataReader** (p. 976) will maintain information regarding a **dds::pub::DataWriter** (p. 1138) once the **dds::pub::DataWriter** (p. 1138) has become not alive.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**lazily_allocate_batch_resources** (p. 1105)

	<<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) pre-allocates resources for receiving batches of samples or lazily allocates them when needed.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_concurrent_blocking_threads** (p. 1238)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The initial number of threads that are allowed to concurrently block on write call on the same **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_concurrent_blocking_threads** (p. 1238)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum number of threads that are allowed to concurrently block on write call on the same **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_remote_reader_filters** (p. 1239)
	<<***extension***>> *(p. 116)* The maximum number of remote DataReaders for which the **dds::pub::DataWriter** (p. 1138) will perform content-based filtering.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_batches** (p. 1239)
	<<***extension***>> *(p. 116)* Represents the initial number of batches a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_batches** (p. 1240)
	<<***extension***>> *(p. 116)* Represents the maximum number of batches a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**instance_replacement** (p. 1241)
	<<***extension***>> *(p. 116)* Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**replace_empty_instances** (p. 1242)
	<<***extension***>> *(p. 116)* Whether or not to replace empty instances during instance replacement
rti::core::policy::DataWriterResourceLimits (p. 1235)	**autoregister_instances** (p. 1242)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Whether or not to automatically register new instances
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_virtual_writers** (p. 1244)
	<<***extension***>> *(p. 116)* The maximum number of virtual writers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_virtual_writers** (p. 1243)

	<<***extension***>> *(*p. *116)* The initial number of virtual writers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_remote_readers** (p. 1244)
	<<***extension***>> *(*p. *116)* Controls the initial memory allocation for the matched remote readers of a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_remote_readers** (p. 1244)
	<<***extension***>> *(*p. *116)* The maximum number of remote readers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_app_ack_remote_readers** (p. 1245)
	<<***extension***>> *(*p. *116)* The maximum number of application-level acknowledging remote readers supported by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_active_topic_queries** (p. 1246)
	<<***extension***>> *(*p. *116)* Represents the initial number of active topic queries a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_active_topic_queries** (p. 1246)
	<<***extension***>> *(*p. *116)* Represents the maximum number of active topic queries a **dds::pub::DataWriter** (p. 1138) will manage.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**writer_loaned_sample_allocation** (p. 1247)
	<<***extension***>> *(*p. *116)* Represents the allocation settings of loaned samples managed by a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initialize_writer_loaned_sample** (p. 1248)
	<<***extension***>> *(*p. *116)* Whether or not to initialize loaned samples returned by a **dds::pub::DataWriter** (p. 1138)
rti::core::policy::DataWriterTransferMode (p. 1253)	**shmem_ref_settings** (p. 1254)
	<<***extension***>> *(*p. *116)* Settings related to transferring data using shared memory references.
rti::core::policy::DataReaderProtocol (p. 1059)	**virtual_guid** (p. 1060)
	<<***extension***>> *(*p. *116)* The virtual GUID (Global Unique Identifier).
rti::core::policy::DataReaderProtocol (p. 1059)	**rtps_object_id** (p. 1061)
	<<***extension***>> *(*p. *116)* The RTPS Object ID.

rti::core::policy::DataReaderProtocol (p. 1059)	**rtps_reliable_reader** (p. 1064)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **dds::sub::DataReader** (p. 976). This parameter only has effect if the reader is configured with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core::policy::ReliabilityKind_def** (p. 2156).
rti::core::policy::DataReaderProtocol (p. 1059)	**expects_inline_qos** (p. 1062)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Specifies whether this DataReader expects inline QoS with every sample.
rti::core::policy::DataReaderProtocol (p. 1059)	**disable_positive_acks** (p. 1062)
	<<***extension***>> *(p. 116)* Whether the reader sends positive acknowledgements to writers.
rti::core::policy::DataReaderProtocol (p. 1059)	**propagate_dispose_of_unregistered_instances** (p. 1063)
	<<***extension***>> *(p. 116)* Indicates whether or not an instance can move to the **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) state without being in the **dds::sub::status::InstanceState::alive()** (p. 1617) state.
rti::core::policy::DataReaderProtocol (p. 1059)	**propagate_unregister_of_disposed_instances** (p. 1064)
	<<***extension***>> *(p. 116)* Indicates whether or not an instance can move to the **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) state directly from the **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617).
rti::core::policy::DataWriterProtocol (p. 1210)	**virtual_guid** (p. 1212)
	<<***extension***>> *(p. 116)* The virtual GUID (Global Unique Identifier).
rti::core::policy::DataWriterProtocol (p. 1210)	**rtps_object_id** (p. 1212)
	<<***extension***>> *(p. 116)* The RTPS Object ID.
rti::core::policy::DataWriterProtocol (p. 1210)	**push_on_write** (p. 1213)
	<<***extension***>> *(p. 116)* Whether to push sample out when write is called.
rti::core::policy::DataWriterProtocol (p. 1210)	**disable_positive_acks** (p. 1214)
	<<***extension***>> *(p. 116)* Controls whether or not the writer expects positive acknowledgements from matching readers.
rti::core::policy::DataWriterProtocol (p. 1210)	**disable_inline_keyhash** (p. 1214)

	<<***extension***>> *(p. 116)* Controls whether or not a keyhash is propagated on the wire with each sample.
rti::core::policy::DataWriterProtocol (p. 1210)	**serialize_key_with_dispose** (p. 1215)
	<<***extension***>> *(p. 116)* Controls whether or not the serialized key is propagated on the wire with dispose samples.
rti::core::policy::DataWriterProtocol (p. 1210)	**propagate_app_ack_with_no_response** (p. 1216)
	<<***extension***>> *(p. 116)* Controls whether or not a **dds::pub::DataWriter** (p. 1138) receives **dds::pub::DataWriterListener::on_application_acknowledgment** (p. 1207) notifications with an empty or invalid response.
rti::core::policy::DataWriterProtocol (p. 1210)	**rtps_reliable_writer** (p. 1217)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **dds::pub::DataWriter** (p. 1138). This parameter only has effect if both the writer and the matching reader are configured with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core::policy::ReliabilityKind_def** (p. 2156).
rti::core::policy::DataWriterProtocol (p. 1210)	**initial_virtual_sequence_number** (p. 1218)
	<<***extension***>> *(p. 116)* Determines, the initial virtual sequence number for this DataWriter.
rti::core::policy::SystemResourceLimits (p. 2443)	**max_objects_per_thread** (p. 2445)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum number of objects that can be stored per thread for a DomainParticipantFactory.
rti::core::policy::SystemResourceLimits (p. 2443)	**initial_objects_per_thread** (p. 2445)
	<<***extension***>> *(p. 116)* The number of objects per thread for a DomainParticipantFactory for which infrastructure will initially be allocated.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_writer_allocation** (p. 1398)
	<<***extension***>> *(p. 116)* Allocation settings applied to local DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_reader_allocation** (p. 1398)
	<<***extension***>> *(p. 116)* Allocation settings applied to local DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_publisher_allocation** (p. 1399)

	<<***extension***>> *(p. 116)* Allocation settings applied to local Publisher.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_subscriber_allocation** (p. 1399)
	<<***extension***>> *(p. 116)* Allocation settings applied to local Subscriber.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_topic_allocation** (p. 1399)
	<<***extension***>> *(p. 116)* Allocation settings applied to local Topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_writer_allocation** (p. 1400)
	<<***extension***>> *(p. 116)* Allocation settings applied to remote DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_reader_allocation** (p. 1401)
	<<***extension***>> *(p. 116)* Allocation settings applied to remote DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_participant_allocation** (p. 1401)
	<<***extension***>> *(p. 116)* Allocation settings applied to remote DomainParticipants.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_writer_reader_pair_allocation** (p. 1402)
	<<***extension***>> *(p. 116)* Allocation settings applied to matching local writer and remote/local reader pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_reader_writer_pair_allocation** (p. 1402)
	<<***extension***>> *(p. 116)* Allocation settings applied to matching local reader and remote/local writer pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**ignored_entity_allocation** (p. 1403)
	<<***extension***>> *(p. 116)* Allocation settings applied to ignored entities.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filtered_topic_allocation** (p. 1403)
	<<***extension***>> *(p. 116)* Allocation settings applied to content filtered topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filter_allocation** (p. 1404)

	<<***extension***>> *(p. 116)* Allocation settings applied to content filter.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**read_condition_allocation** (p. 1404)
	<<***extension***>> *(p. 116)* Allocation settings applied to read condition pool.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**query_condition_allocation** (p. 1405)
	<<***extension***>> *(p. 116)* Allocation settings applied to query condition pool.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**outstanding_asynchronous_sample_allocation** (p. 1405)
	<<***extension***>> *(p. 116)* Allocation settings applied to the maximum number of samples (from all **dds::pub::↩ DataWriter** (p. 1138)) waiting to be asynchronously written.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**flow_controller_allocation** (p. 1406)
	<<***extension***>> *(p. 116)* Allocation settings applied to flow controllers.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_writer_hash_buckets** (p. 1406)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Hash_Buckets settings applied to local DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_reader_hash_buckets** (p. 1407)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_publisher_hash_buckets** (p. 1407)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local Publisher.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_subscriber_hash_buckets** (p. 1408)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local Subscriber.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_topic_hash_buckets** (p. 1408)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local Topic.

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_writer_hash_buckets** (p. 1409)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_reader_hash_buckets** (p. 1409)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_participant_hash_buckets** (p. 1410)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote DomainParticipants.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_writer_reader_pair_hash_buckets** (p. 1410)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for matching local writer and remote/local reader pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_reader_writer_pair_hash_buckets** (p. 1411)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for matching local reader and remote/local writer pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**ignored_entity_hash_buckets** (p. 1411)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for ignored entities.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filtered_topic_hash_buckets** (p. 1412)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for content filtered topics.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filter_hash_buckets** (p. 1412)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for content filters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**flow_controller_hash_buckets** (p. 1413)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for flow controllers.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_gather_destinations** (p. 1413)
	<<***extension***>> *(p. 116)* Maximum number of destinations per RTI Connext send.

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**participant_user_data_max_length** (p. 1414)
	<<***extension***>> *(p. 116)* Maximum length of user data in **dds::domain::qos::DomainParticipantQos** (p. 1384) and **dds::topic::ParticipantBuiltinTopicData** (p. 1908)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**topic_data_max_length** (p. 1414)
	<<***extension***>> *(p. 116)* Maximum length of topic data in **dds::topic::qos::TopicQos** (p. 2509), **dds::topic::TopicBuiltinTopicData** (p. 2493), **dds::topic::PublicationBuiltinTopicData** (p. 1977) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**publisher_group_data_max_length** (p. 1415)
	<<***extension***>> *(p. 116)* Maximum length of group data in **dds::pub::qos::PublisherQos** (p. 2007) and **dds::topic::PublicationBuiltinTopicData** (p. 1977)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**subscriber_group_data_max_length** (p. 1415)
	<<***extension***>> *(p. 116)* Maximum length of group data in **dds::sub::qos::SubscriberQos** (p. 2423) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_user_data_max_length** (p. 1416)
	<<***extension***>> *(p. 116)* Maximum length of user data in **dds::pub::qos::DataWriterQos** (p. 1226) and **dds::topic::PublicationBuiltinTopicData** (p. 1977)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_user_data_max_length** (p. 1416)
	<<***extension***>> *(p. 116)* Maximum length of user data in **dds::sub::qos::DataReaderQos** (p. 1072) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_partitions** (p. 1417)
	<<***extension***>> *(p. 116)* Maximum number of partition name strings allowable in a **dds::core::policy::Partition** (p. 1923).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_partition_cumulative_characters** (p. 1417)
	<<***extension***>> *(p. 116)* Maximum number of combined characters allowable in all partition names in a **dds::core::policy::Partition** (p. 1923).

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_code_max_serialized_length** (p. 1418)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Maximum size of serialized string for type code.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_object_max_serialized_length** (p. 1419)
	<<***extension***>> *(p. 116)* The maximum length, in bytes, that the buffer to serialize a TypeObject v1 can consume.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_object_max_deserialized_length** (p. 1420)
	<<***extension***>> *(p. 116)* The maximum number of bytes that a deserialized TypeObject v1 can consume.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**deserialized_type_object_dynamic_allocation_threshold** (p. 1420)
	<<***extension***>> *(p. 116)* A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject v1.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**serialized_type_object_dynamic_allocation_threshold** (p. 1421)
	<<***extension***>> *(p. 116)* A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject v1.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**contentfilter_property_max_length** (p. 1422)
	<<***extension***>> *(p. 116)* This field is the maximum length of all data related to a Content-filtered topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**channel_seq_max_length** (p. 1423)
	<<***extension***>> *(p. 116)* Maximum number of channels that can be specified in **rti::core::policy::MultiChannel** (p. 1749) for MultiChannel DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**channel_filter_expression_max_length** (p. 1424)
	<<***extension***>> *(p. 116)* Maximum length of a channel **rti::core::ChannelSettings::filter_expression** (p. 923) in a MultiChannel DataWriter.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**participant_property_list_max_length** (p. 1424)
	<<***extension***>> *(p. 116)* Maximum number of properties associated with the **dds::domain::DomainParticipant** (p. 1326).

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**participant_property_string_max_length** (p. 1425)
	<<***extension***>> *(*p. *116)* Maximum string length of the properties associated with the **dds::domain::Domain**↩ **Participant** (p. 1326).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_property_list_max_length** (p. 1425)
	<<***extension***>> *(*p. *116)* Maximum number of properties associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_property_string_max_length** (p. 1426)
	<<***extension***>> *(*p. *116)* Maximum string length of the properties associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_property_list_max_length** (p. 1426)
	<<***extension***>> *(*p. *116)* Maximum number of properties associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_property_string_max_length** (p. 1427)
	<<***extension***>> *(*p. *116)* Maximum string length of the properties associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_endpoint_groups** (p. 1427)
	<<***extension***>> *(*p. *116)* Maximum number of **rti**↩ **::core::EndpointGroup** (p. 1515) allowable in a **rti**↩ **::core::policy::Availability** (p. 868).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**max_endpoint_group_cumulative_characters** (p. 1428)
	<<***extension***>> *(*p. *116)* Maximum number of combined role_name characters allowed in all **rti::core**↩ **::EndpointGroup** (p. 1515) in a **rti::core::policy::**↩ **Availability** (p. 868).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**transport_info_list_max_length** (p. 1428)
	<<***extension***>> *(*p. *116)* Maximum number of installed transports to send and receive information about in **dds::topic::ParticipantBuiltinTopic**↩ **Data::transport_info** (p. 1914).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**ignored_entity_replacement_kind** (p. 1429)

	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **rti::core::policy::DomainParticipant↩ ResourceLimits::ignored_entity_allocation** (p. 1403) are reached.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_topic_query_allocation** (p. 1429)
	<<***extension***>> *(p. 116)* Allocation settings applied to remote TopicQueries.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_topic_query_hash_buckets** (p. 1430)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote TopicQueries.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_data_tag_list_max_length** (p. 1431)
	<<***extension***>> *(p. 116)* Maximum number of data tags associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**writer_data_tag_string_max_length** (p. 1431)
	<<***extension***>> *(p. 116)* Maximum string length of the data tags associated with a **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_data_tag_list_max_length** (p. 1432)
	<<***extension***>> *(p. 116)* Maximum number of data tags associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**reader_data_tag_string_max_length** (p. 1432)
	<<***extension***>> *(p. 116)* Maximum string length of the data tags associated with a **dds::sub::DataReader** (p. 976).
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**shmem_ref_transfer_mode_max_segments** (p. 1433)
	<<***extension***>> *(p. 116)* Maximum number of segments created by all DataWriters belonging to a **dds↩ ::domain::DomainParticipant** (p. 1326).
rti::core::policy::Event (p. 1541)	**thread** (p. 1542)
	<<***extension***>> *(p. 116)* Event thread QoS.
rti::core::policy::Event (p. 1541)	**initial_count** (p. 1543)
	<<***extension***>> *(p. 116)* The initial number of events.

rti::core::policy::Event (p. 1541)	**max_count** (p. 1543)
	<<***extension***>> *(p. 116)* The maximum number of events.
rti::core::policy::Database (p. 969)	**thread** (p. 971)
	<<***extension***>> *(p. 116)* Database thread settings.
rti::core::policy::Database (p. 969)	**shutdown_timeout** (p. 972)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum wait time during a shutdown.
rti::core::policy::Database (p. 969)	**cleanup_period** (p. 972)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The period at which the database thread wakes up to remove deleted records.
rti::core::policy::Database (p. 969)	**shutdown_cleanup_period** (p. 973)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The clean-up period used during database shut-down.
rti::core::policy::Database (p. 969)	**initial_records** (p. 974)
	<<***extension***>> *(p. 116)* The initial number of total records.
rti::core::policy::Database (p. 969)	**max_skiplist_level** (p. 974)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum level of the skiplist.
rti::core::policy::Database (p. 969)	**initial_weak_references** (p. 975)
	<<***extension***>> *(p. 116)* The initial number of weak references.
rti::core::policy::Database (p. 969)	**max_weak_references** (p. 974)
	<<***extension***>> *(p. 116)* The maximum number of weak references.
rti::core::policy::ReceiverPool (p. 2144)	**thread** (p. 2146)
	<<***extension***>> *(p. 116)* Receiver pool thread(s).
rti::core::policy::ReceiverPool (p. 2144)	**buffer_size** (p. 2147)
	<<***extension***>> *(p. 116)* The receive buffer size in bytes.
rti::core::policy::ReceiverPool (p. 2144)	**buffer_alignment** (p. 2148)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The receive buffer alignment.
rti::core::policy::DiscoveryConfig (p. 1271)	**remote_participant_purge_kind** (p. 1281)

	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
rti::core::policy::DiscoveryConfig (p. 1271)	**max_liveliness_loss_detection_period** (p. 1281)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_reader_resource_limits** (p. 1285)
	<<***extension***>> *(p. 116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_reader** (p. 1286)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_reader_resource_limits** (p. 1287)
	<<***extension***>> *(p. 116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_reader** (p. 1288)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_reader_resource_limits** (p. 1289)
	<<***extension***>> *(p. 116)* Resource limits.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_writer** (p. 1296)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core::policy::ReliabilityKind_def** (p. 2156).
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_writer_publish_mode** (p. 1298)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in publication writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_writer_publish_mode** (p. 1298)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in subscription writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**asynchronous_publisher** (p. 1299)

	<<***extension***>> *(*p. *116)* Asynchronous publishing settings for the discovery **dds::pub::Publisher** (p. 1993) and all entities that are created by it.
rti::core::policy::DiscoveryConfig (p. 1271)	**default_domain_announcement_period** (p. 1300)
	<<***extension***>> *(*p. *116)* The period to announce a participant to the default domain 0.
rti::core::policy::DiscoveryConfig (p. 1271)	**ignore_default_domain_announcements** (p. 1301)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_reader_reliability_kind** (p. 1295)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Reliability policy for a built-in participant message reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_reader** (p. 1295)
	<<***extension***>> *(*p. *116)* RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **rti::core::policy::DiscoveryConfig::participant_message_reader_reliability_kind** (p. 1295) is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157).
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_writer** (p. 1289)
	<<***extension***>> *(*p. *116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**publication_writer_data_lifecycle** (p. 1291)
	<<***extension***>> *(*p. *116)* Writer data lifecycle settings for a built-in publication writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_writer** (p. 1291)
	<<***extension***>> *(*p. *116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**subscription_writer_data_lifecycle** (p. 1293)
	<<***extension***>> *(*p. *116)* Writer data lifecycle settings for a built-in subscription writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**builtin_discovery_plugins** (p. 1293)
	<<***extension***>> *(*p. *116)* Mask of built-in discovery plugin kinds.

rti::core::policy::DiscoveryConfig (p. 1271)	**enabled_builtin_channels** (p. 1294)
	<<***extension***>> *(p. 116)* The mask specifying which built-in channels should be enabled.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_writer** (p. 1301)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **rti::topic::ServiceRequest** (p. 2358) writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_writer_data_lifecycle** (p. 1303)
	<<***extension***>> *(p. 116)* Writer data lifecycle settings for a built-in **rti::topic::ServiceRequest** (p. 2358) writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_writer_publish_mode** (p. 1303)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in service request writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**service_request_reader** (p. 1304)
	<<***extension***>> *(p. 116)* RTPS reliable reader protocol-related configuration settings for a built-in **rti::topic::ServiceRequest** (p. 2358) reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**locator_reachability_assert_period** (p. 1305)
	<<***extension***>> *(p. 116)* Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.
rti::core::policy::DiscoveryConfig (p. 1271)	**locator_reachability_change_detection_period** (p. 1306)
	<<***extension***>> *(p. 116)* Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_writer** (p. 1307)
	<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_writer_publish_mode** (p. 1309)
	<<***extension***>> *(p. 116)* Publish mode policy for the built-in secure volatile writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**secure_volatile_reader** (p. 1309)
	<<***extension***>> *(p. 116)* RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**endpoint_type_object_lb_serialization_threshold** (p. 1310)

	<<***extension***>> *(*p. *116)* Controls when/if the Type↩ Object v1 is compressed when propagated in Simple Endpoint Discovery.
rti::core::policy::DiscoveryConfig (p. 1271)	**dns_tracker_polling_period** (p. 1311)
	<<***extension***>> *(*p. *116)* Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_reader** (p. 1312)
	<<***extension***>> *(*p. *116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_reader_resource_limits** (p. 1313)
	<<***extension***>> *(*p. *116)* Resource limits for the built-in topic participant configuration reader.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_writer_publish_mode** (p. 1316)
	<<***extension***>> *(*p. *116)* Publish mode policy for the built-in participant configuration writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_writer** (p. 1314)
	<<***extension***>> *(*p. *116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_configuration_writer_data_lifecycle** (p. 1316)
	<<***extension***>> *(*p. *116)* Writer data lifecycle settings for a built-in participant configuration writer.
rti::core::policy::DiscoveryConfig (p. 1271)	**request_types_filter** (p. 1317)
	<<***extension***>> *(*p. *116)* Topic name pattern for requesting all types via Type Lookup Service.
rti::core::RtpsReliableReaderProtocol (p. 2221)	**min_heartbeat_response_delay** (p. 2223)
	<<***extension***>> *(*p. *116)* The minimum delay to respond to a heartbeat.
rti::core::RtpsReliableReaderProtocol (p. 2221)	**max_heartbeat_response_delay** (p. 2224)
	<<***extension***>> *(*p. *116)* The maximum delay to respond to a heartbeat.
rti::core::RtpsReliableReaderProtocol (p. 2221)	**heartbeat_suppression_duration** (p. 2224)
	<<***extension***>> *(*p. *116)* The duration a reader ignores consecutively received heartbeats

rti::core::RtpsReliableReaderProtocol (p. 2221)	**nack_period** (p. 2225)
	<<***extension***>> *(p. 116)* The period at which to send NACKs
rti::core::RtpsReliableReaderProtocol (p. 2221)	**receive_window_size** (p. 2226)
	<<***extension***>> *(p. 116)* The number of received out-of-order samples a reader can keep at a time
rti::core::RtpsReliableReaderProtocol (p. 2221)	**round_trip_time** (p. 2226)
	<<***extension***>> *(p. 116)* The duration from sending a NACK to receiving a repair of a sample
rti::core::RtpsReliableReaderProtocol (p. 2221)	**app_ack_period** (p. 2227)
	<<***extension***>> *(p. 116)* The period at which application-level acknowledgment messages are sent
rti::core::RtpsReliableReaderProtocol (p. 2221)	**min_app_ack_response_keep_duration** (p. 2227)
	<<***extension***>> *(p. 116)* Minimum duration for which application-level acknowledgment response data is kept
rti::core::RtpsReliableReaderProtocol (p. 2221)	**samples_per_app_ack** (p. 2228)
	<<***extension***>> *(p. 116)* The minimum number of samples acknowledged by one application-level acknowledgment message
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**low_watermark** (p. 2232)
	<<***extension***>> *(p. 116)* When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385) is considered to have changed.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**high_watermark** (p. 2233)
	<<***extension***>> *(p. 116)* When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, the **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385) is considered to have changed.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**virtual_heartbeat_period** (p. 2237)
	<<***extension***>> *(p. 116)* The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.

rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**samples_per_virtual_heartbeat** (p. 2238)
	<<***extension***>> *(*p. *116)* The number of samples that a reliable writer has to publish before sending a virtual heartbeat.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_heartbeat_retries** (p. 2239)
	<<***extension***>> *(*p. *116)* The maximum number of *periodic* heartbeat retries before marking a remote reader as inactive.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**inactivate_nonprogressing_readers** (p. 2239)
	<<***extension***>> *(*p. *116)* Whether to treat remote readers as inactive when their NACKs do not progress
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**heartbeats_per_max_samples** (p. 2240)
	<<***extension***>> *(*p. *116)* The number of piggyback heartbeats sent per max send window.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**min_nack_response_delay** (p. 2242)
	<<***extension***>> *(*p. *116)* The minimum delay to respond to a NACK or NACK_FRAG.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_nack_response_delay** (p. 2242)
	<<***extension***>> *(*p. *116)* The maximum delay to respond to a NACK or NACK_FRAG.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**nack_suppression_duration** (p. 2243)
	<<***extension***>> *(*p. *116)* The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**max_bytes_per_nack_response** (p. 2244)
	<<***extension***>> *(*p. *116)* The maximum total message size when resending rejected samples or data fragments.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_enable_adaptive_sample_↩ keep_duration** (p. 2247)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Enables dynamic adjustment of sample keep duration in response to congestion.

rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_min_sample_keep_duration** (p. 2245)
	<<***extension***>> *(p. 116)* The minimum duration a sample is queued for ACK-disabled readers.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_max_sample_keep_duration** (p. 2246)
	<<***extension***>> *(p. 116)* The maximum duration a sample is queued for ACK-disabled readers.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_decrease_sample_keep_duration_factor** (p. 2248)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Controls rate of contraction of dynamic sample keep duration.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_increase_sample_keep_duration_factor** (p. 2248)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Controls rate of growth of dynamic sample keep duration.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**send_window_update_period** (p. 2251)
	<<***extension***>> *(p. 116)* Period in which send window may be dynamically changed
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**send_window_increase_factor** (p. 2252)
	<<***extension***>> *(p. 116)* Increases send window size by this percentage when reacting dynamically to network conditions
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**send_window_decrease_factor** (p. 2253)
	<<***extension***>> *(p. 116)* Decreases send window size by this percentage when reacting dynamically to network conditions
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**enable_multicast_periodic_heartbeat** (p. 2254)
	<<***extension***>> *(p. 116)* Whether periodic heartbeat messages are sent over multicast
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**multicast_resend_threshold** (p. 2254)
	<<***extension***>> *(p. 116)* The minimum number of requesting readers needed to trigger a multicast resend
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_repair_piggyback_heartbeat** (p. 2255)

	<<***extension***>> *(*p. *116)* Prevents piggyback heartbeats from being sent with repair samples
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_samples** (p. 909)
	<<***extension***>> *(*p. *116)* Initial number of samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_samples** (p. 909)
	<<***extension***>> *(*p. *116)* Maximum number of samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_infos** (p. 910)
	<<***extension***>> *(*p. *116)* Initial number of sample infos.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_infos** (p. 911)
	<<***extension***>> *(*p. *116)* Maximum number of sample infos.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_outstanding_reads** (p. 911)
	<<***extension***>> *(*p. *116)* The initial number of outstanding reads that have not called finish yet on the same built-in topic **dds::sub::DataReader** (p. 976).
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_outstanding_reads** (p. 912)
	<<***extension***>> *(*p. *116)* The maximum number of outstanding reads that have not called finish yet on the same built-in topic **dds::sub::DataReader** (p. 976).
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_samples_per_read** (p. 912)
	<<***extension***>> *(*p. *116)* Maximum number of samples that can be read/taken on a same built-in topic **dds::sub::DataReader** (p. 976).
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**disable_fragmentation_support** (p. 913)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Determines whether the built-in topic **dds::sub::Data↩ Reader** (p. 976) can receive fragmented samples.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragmented_samples** (p. 913)

	<<***extension***>> *(*p. *116)* The maximum number of samples for which the built-in topic **dds::sub::Data↩ Reader** (p. 976) may store fragments at a given point in time.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**initial_fragmented_samples** (p. 914)
	<<***extension***>> *(*p. *116)* The initial number of samples for which a built-in topic **dds::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragmented_samples_per_remote_writer** (p. 915)
	<<***extension***>> *(*p. *116)* The maximum number of samples per remote writer for which a built-in topic **dds↩ ::sub::DataReader** (p. 976) may store fragments.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**max_fragments_per_sample** (p. 916)
	<<***extension***>> *(*p. *116)* Maximum number of fragments for a single sample.
rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**dynamically_allocate_fragmented_samples** (p. 916)
	<<***extension***>> *(*p. *116)* Determines whether the built-in topic **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.
dds::core::policy::DataTag (p. 1132)	**tags** (p. 1137)
	<<***spec***>> *(*p. *112)* Sequence of data tags.
rti::core::policy::EntityName (p. 1530)	**role_name** (p. 1532)
	<<***extension***>> *(*p. *116)* The entity role name.
rti::core::policy::Service (p. 2350)	**kind** (p. 2354)
	<<***extension***>> *(*p. *116)* The kind of service.
rti::core::policy::Batch (p. 880)	**max_flush_delay** (p. 884)
	<<***extension***>> *(*p. *116)* The maximum flush delay.
rti::core::policy::Batch (p. 880)	**source_timestamp_resolution** (p. 885)
	<<***extension***>> *(*p. *116)* Batch source timestamp resolution.
rti::core::policy::Batch (p. 880)	**thread_safe_write** (p. 886)
	<<***extension***>> *(*p. *116)* Determines whether or not the write operation is thread safe.
rti::core::ChannelSettings (p. 921)	**multicast_settings** (p. 922)

	<<***extension***>> *(*p. *116)* A sequence of **rti::core::↩ TransportMulticastSettings** (p. 2549) used to configure the multicast addresses associated with a channel.
rti::core::ChannelSettings (p. 921)	**filter_expression** (p. 923)
	<<***extension***>> *(*p. *116)* A logical expression used to determine the data that will be published in the channel.
rti::core::ChannelSettings (p. 921)	**priority** (p. 924)
	<<***extension***>> *(*p. *116)* Publication priority
rti::core::policy::MultiChannel (p. 1749)	**channels** (p. 1751)
	<<***extension***>> *(*p. *116)* A sequence of **rti::core::↩ ChannelSettings** (p. 921) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.
rti::core::policy::MultiChannel (p. 1749)	**filter_name** (p. 1752)
	<<***extension***>> *(*p. *116)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.
rti::core::policy::TopicQueryDispatch (p. 2522)	**enable** (p. 2524)
	<<***extension***>> *(*p. *116)* Allows this writer to dispatch TopicQueries
rti::core::policy::TopicQueryDispatch (p. 2522)	**samples_per_period** (p. 2525)
	<<***extension***>> *(*p. *116)* Sets the maximum number of samples to publish in each publication_period.

6.77 Mutable QoS

This page lists all QoS that are always mutable.

This page lists all QoS that are always mutable.

Mutable QoS can always be changed regardless of whether the entity has been enabled or not. QoS that are immutable after an entity has been enabled are listed on the **Immutable QoS** (p. 243) page.

Basic Settings

dds::core::policy::Deadline (p. 1255)	**period** (p. 1258)
	<<***basic***>> *(*p. *104)* <<***spec***>> *(*p. *112)* Duration of the deadline period.
dds::core::policy::OwnershipStrength (p. 1906)	**value** (p. 1908)

	<<***basic***>> (p. *104*) <<***spec***>> (p. *112*) The strength value used to arbitrate among multiple writers.
dds::core::policy::TimeBasedFilter (p. 2469)	**minimum_separation** (p. 2472)
	<<***basic***>> (p. *104*) <<***spec***>> (p. *112*) The minimum separation duration between subsequent samples.
dds::core::policy::Partition (p. 1923)	**name** (p. 1926)
	<<***basic***>> (p. *104*) <<***spec***>> (p. *112*) A list of partition names
dds::core::policy::EntityFactory (p. 1527)	**autoenable_created_entities** (p. 1529)
	<<***basic***>> (p. *104*) <<***spec***>> (p. *112*) Specifies whether the entity acting as a factory automatically enables the instances it creates.
dds::core::policy::Lifespan (p. 1635)	**duration** (p. 1637)
	<<***basic***>> (p. *104*) <<***spec***>> (p. *112*) Maximum duration for the data's validity.
rti::core::policy::Property (p. 1969)	**value** (p. 1973)
	<<***basic***>> (p. *104*) <<***extension***>> (p. *116*) Sequence of properties.
rti::config::Logger (p. 1683)	**verbosity** (p. 1685)
	<<***basic***>> (p. *104*) <<***extension***>> (p. *116*) The verbosities at which RTI Connext diagnostic information is logged.
rti::config::Logger (p. 1683)	**category** (p. 1685)
	<<***basic***>> (p. *104*) <<***extension***>> (p. *116*) Categories of logged messages.
rti::config::Logger (p. 1683)	**output_file** (p. 1686)
	<<***basic***>> (p. *104*) <<***extension***>> (p. *116*) Specifies the file to which log messages will be redirected to.
rti::config::Logger (p. 1683)	**output_file_suffix** (p. 1687)
	<<***basic***>> (p. *104*) <<***extension***>> (p. *116*) Sets the file suffix when logging to a set of files
rti::core::MonitoringEventDistributionSettings (p. 1722)	**publication_period** (p. 1726)
	<<***basic***>> (p. *104*) <<***extension***>> (p. *116*) Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**polling_period** (p. 1744)

	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Period at which the periodic metric thread polls and publishes the periodic metrics.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**max_historical_logs** (p. 1729)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**publication_period** (p. 1732)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Period at which the logging thread publishes log messages.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**enable** (p. 1713)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Enables the use of a dedicated **dds::domain::Domain↩ Participant** (p. 1326) to distribute the RTI Connext application telemetry data.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**domain_id** (p. 1713)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* The domain ID used in the creation of RTI Monitoring Library 2.0 **dds::domain::DomainParticipant** (p. 1326).
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**collector_initial_peers** (p. 1715)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **rti::core::MonitoringDistribution↩ Settings::dedicated_participant** (p. 1719).
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**middleware_forwarding_level** (p. 1734)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Log messages with **rti::config::LogFacility::middleware** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**security_event_forwarding_level** (p. 1734)
	<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Log messages with **rti::config::LogFacility::security_↩ event** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**service_forwarding_level** (p. 1735)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Log messages with **rti::config::LogFacility::service** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringLoggingForwardingSettings (p. 1733)	**user_forwarding_level** (p. 1736)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Log messages with **rti::config::LogFacility::user** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
rti::core::MonitoringTelemetryData (p. 1744)	**metrics** (p. 1745)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Sequence of **rti::core::MonitoringMetricSelection** (p. 1736) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.
rti::core::MonitoringTelemetryData (p. 1744)	**logs** (p. 1746)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* **rti::core::MonitoringLoggingForwardingSettings** (p. 1733) containing the **rti::config::SyslogVerbosity** (p. 458) levels that will be forwarded for the different **rti::config::LogFacility** (p. 459).
rti::core::policy::Monitoring (p. 1706)	**enable** (p. 1708)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.
rti::core::policy::Monitoring (p. 1706)	**application_name** (p. 1709)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The name of the resource that represents this RTI Connext application.

Advanced Settings

dds::core::policy::UserData (p. 2591)	**value** (p. 2593)
	<<***spec***>> *(p. 112)* a sequence of octets
dds::core::policy::GroupData (p. 1591)	**value** (p. 1593)
	<<***spec***>> *(p. 112)* a sequence of octets
rti::core::policy::Availability (p. 868)	**max_data_availability_waiting_time** (p. 871)

	<<***extension***>> *(p. 116)* Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
rti::core::policy::Availability (p. 868)	**max_endpoint_availability_waiting_time** (p. 872)
	<<***extension***>> *(p. 116)* Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
rti::core::policy::Availability (p. 868)	**required_matched_endpoint_groups** (p. 873)
	<<***extension***>> *(p. 116)* A sequence of endpoint groups.
dds::core::policy::LatencyBudget (p. 1631)	**duration** (p. 1633)
	<<***spec***>> *(p. 112)* Duration of the maximum acceptable delay.
rti::config::Logger (p. 1683)	**print_format** (p. 1688)
	<<***extension***>> *(p. 116)* The format used to output RTI Connext diagnostic information.
rti::config::Logger (p. 1683)	**max_bytes_per_file** (p. 1687)
	<<***extension***>> *(p. 116)* Specifies the maximum number of bytes a single file can contain
rti::config::Logger (p. 1683)	**max_files** (p. 1687)
	<<***extension***>> *(p. 116)* Specifies the maximum number of files to create before overwritting the previous ones.
rti::core::MonitoringEventDistributionSettings (p. 1722)	**concurrency_level** (p. 1723)
	<<***extension***>> *(p. 116)* Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
rti::core::MonitoringEventDistributionSettings (p. 1722)	**datawriter_qos_profile_name** (p. 1724)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes event metrics.
rti::core::MonitoringEventDistributionSettings (p. 1722)	**thread** (p. 1726)
	<<***extension***>> *(p. 116)* The settings of the event metric thread.
rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**datawriter_qos_profile_name** (p. 1742)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes periodic metrics.

rti::core::MonitoringPeriodicDistributionSettings (p. 1740)	**thread** (p. 1743)
	<<***extension***>> *(p. 116)* The settings of the periodic metric thread.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**concurrency_level** (p. 1729)
	<<***extension***>> *(p. 116)* Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**datawriter_qos_profile_name** (p. 1730)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes log messages.
rti::core::MonitoringLoggingDistributionSettings (p. 1727)	**thread** (p. 1731)
	<<***extension***>> *(p. 116)* The settings of the logging thread.
rti::core::MonitoringDedicatedParticipantSettings (p. 1712)	**participant_qos_profile_name** (p. 1714)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the **dds::domain::DomainParticipant** (p. 1326) that will be used to distribute telemetry data.
rti::core::MonitoringDistributionSettings (p. 1716)	**dedicated_participant** (p. 1719)
	<<***extension***>> *(p. 116)* Configures the use of a dedicated **dds::domain::DomainParticipant** (p. 1326) to distribute the RTI Connext application telemetry data.
rti::core::MonitoringDistributionSettings (p. 1716)	**publisher_qos_profile_name** (p. 1718)
	<<***extension***>> *(p. 116)* The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.
rti::core::MonitoringDistributionSettings (p. 1716)	**event_settings** (p. 1720)
	<<***extension***>> *(p. 116)* Configures the distribution of event metrics.
rti::core::MonitoringDistributionSettings (p. 1716)	**periodic_settings** (p. 1720)
	<<***extension***>> *(p. 116)* Configures the distribution of periodic metrics.
rti::core::MonitoringDistributionSettings (p. 1716)	**logging_settings** (p. 1721)
	<<***extension***>> *(p. 116)* Configures the distribution of logging messages.

rti::core::MonitoringMetricSelection (p. 1736)	**resource_selection** (p. 1737)
	<<***extension***>> *(p. 116)* An expression pattern that selects a subset of resources by matching the pattern to the resource names. **rti::core::MonitoringMetric↩ Selection::enabled_metrics_selection** (p. 1738) and **rti::core::MonitoringMetricSelection::disabled_↩ metrics_selection** (p. 1739) are applied to this subset of resources.
rti::core::MonitoringMetricSelection (p. 1736)	**enabled_metrics_selection** (p. 1738)
	<<***extension***>> *(p. 116)* A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by **rti::core::Monitoring↩ MetricSelection::resource_selection** (p. 1737).
rti::core::MonitoringMetricSelection (p. 1736)	**disabled_metrics_selection** (p. 1739)
	<<***extension***>> *(p. 116)* A sequence of POSIX fnmatch patterns that mach the names of the metrics that should not be collected and distributed for the observable resources selected by **rti::core::Monitoring↩ MetricSelection::resource_selection** (p. 1737).
rti::core::policy::Monitoring (p. 1706)	**distribution_settings** (p. 1710)
	<<***extension***>> *(p. 116)* Configures the distribution of telemetry data.
rti::core::policy::Monitoring (p. 1706)	**telemetry_data** (p. 1711)
	<<***extension***>> *(p. 116)* Configures the telemetry data that will be distributed.

6.78 Other QoS Attributes

These attributes are characteristics of the QoS – for example if they are **mutable** (p. 280) or **immutable** (p. 243), or if they **apply per-instance** (p. 287).

Modules

- **Applied Per-Instance**

  These QoS are applied per-instance.
- **Discovery Metadata**

  These QoS are sent as part of one of the built-in discovery topics.
- **Matching**

  These QoS affect participant and endpoint matching.

6.78.1 Detailed Description

These attributes are characteristics of the QoS – for example if they are **mutable** (p. 280) or **immutable** (p. 243), or if they **apply per-instance** (p. 287).

6.79 Applied Per-Instance

These QoS are applied per-instance.

These QoS are applied per-instance.

For example, the **dds::core::policy::History::depth** (p. 1605) configures the number of samples that are stored in the DataWriter and DataReader queues *per-instance*.

Basic Settings

dds::core::policy::Durability (p. 1433)	**writer_depth** (p. 1439)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Indicates the number of samples per instance that a durable **dds::pub::↩ DataWriter** (p. 1138) will send to a late-joining **dds::sub::↩ DataReader** (p. 976).
dds::core::policy::Deadline (p. 1255)	**period** (p. 1258)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Duration of the deadline period.
dds::core::policy::Ownership (p. 1899)	**kind** (p. 1904)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* The kind of ownership
dds::core::policy::OwnershipStrength (p. 1906)	**value** (p. 1908)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* The strength value used to arbitrate among multiple writers.
dds::core::policy::TimeBasedFilter (p. 2469)	**minimum_separation** (p. 2472)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* The minimum separation duration between subsequent samples.
dds::core::policy::DestinationOrder (p. 1258)	**kind** (p. 1261)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Specifies the desired kind of destination order.
dds::core::policy::History (p. 1602)	**kind** (p. 1604)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Specifies the kind of history to be kept.
dds::core::policy::History (p. 1602)	**depth** (p. 1605)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Specifies the number of samples per instance to be kept, when the `kind` is **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607)

Advanced Settings

dds::core::policy::DurabilityService (p. 1443)	**history_kind** (p. 1445)
	<<***spec***>> *(p. 112)* The kind of history to apply in recouping durable data.
dds::core::policy::DurabilityService (p. 1443)	**history_depth** (p. 1446)
	<<***spec***>> *(p. 112)* Setting to use for the **dds::core::policy::↩ Durability::writer_depth** (p. 1439) when recouping durable data.
dds::core::policy::Presentation (p. 1942)	**access_scope** (p. 1947)
	<<***spec***>> *(p. 112)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
dds::core::policy::Presentation (p. 1942)	**ordered_access** (p. 1948)
	<<***spec***>> *(p. 112)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
dds::core::policy::DestinationOrder (p. 1258)	**scope** (p. 1262)
	<<***spec***>> *(p. 112)* Specifies the desired scope of the source destination order.
dds::core::policy::DestinationOrder (p. 1258)	**source_timestamp_tolerance** (p. 1262)
	<<***extension***>> *(p. 116)* Allowed tolerance between source timestamps of consecutive samples.

6.80 Discovery Metadata

These QoS are sent as part of one of the built-in discovery topics.

These QoS are sent as part of one of the built-in discovery topics.

See also

dds::topic::ParticipantBuiltinTopicData (p. 1908)

dds::topic::PublicationBuiltinTopicData (p. 1977)

dds::topic::SubscriptionBuiltinTopicData (p. 2428)

Basic Settings

dds::core::policy::Durability (p. 1433)	**kind** (p. 1437)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of durability.
dds::core::policy::DataRepresentation (p. 1113)	**compression_settings** (p. 1117)
	*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Structure that contains the compression settings.
dds::core::policy::Deadline (p. 1255)	**period** (p. 1258)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Duration of the deadline period.
dds::core::policy::Ownership (p. 1899)	**kind** (p. 1904)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of ownership
dds::core::policy::OwnershipStrength (p. 1906)	**value** (p. 1908)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The strength value used to arbitrate among multiple writers.
dds::core::policy::Liveliness (p. 1645)	**kind** (p. 1649)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of liveliness desired.
dds::core::policy::Liveliness (p. 1645)	**lease_duration** (p. 1649)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The duration within which a **dds::pub::DataWriter** (p. 1138) must be asserted, or else it is assumed to be not alive.
dds::core::policy::TimeBasedFilter (p. 2469)	**minimum_separation** (p. 2472)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The minimum separation duration between subsequent samples.
dds::core::policy::Partition (p. 1923)	**name** (p. 1926)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* A list of partition names
dds::core::policy::Reliability (p. 2149)	**kind** (p. 2153)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Kind of reliability.
dds::core::policy::Reliability (p. 2149)	**max_blocking_time** (p. 2153)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The maximum time a DataWriter can block on a **write()** (p. 1179) call.
dds::core::policy::DestinationOrder (p. 1258)	**kind** (p. 1261)
	*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Specifies the desired kind of destination order.

dds::core::policy::Lifespan (p. 1635)	**duration** (p. 1637)
	<<***basic***>> *(p. 104)* <<***spec***>> *(p. 112)* Maximum duration for the data's validity.
rti::core::policy::TransportMulticast (p. 2543)	**value** (p. 2545)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* A sequence of multicast communications settings.
rti::core::policy::Discovery (p. 1265)	**multicast_receive_addresses** (p. 1268)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the DomainParticipant.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_liveliness_lease_duration** (p. 1279)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The liveliness lease duration for the participant.
rti::core::policy::Property (p. 1969)	**value** (p. 1973)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* Sequence of properties.
rti::core::policy::EntityName (p. 1530)	**name** (p. 1531)
	<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The name of the entity.

Advanced Settings

dds::core::policy::UserData (p. 2591)	**value** (p. 2593)
	<<***spec***>> *(p. 112)* a sequence of octets
dds::core::policy::GroupData (p. 1591)	**value** (p. 1593)
	<<***spec***>> *(p. 112)* a sequence of octets
dds::core::policy::Durability (p. 1433)	**direct_communication** (p. 1438)
	<<***extension***>> *(p. 116)* Indicates whether or not a TRANSIENT or PERSISTENT **dds::sub::DataReader** (p. 976) should receive samples directly from a TRANSIENT or PERSISTENT **dds::pub::DataWriter** (p. 1138)
dds::core::policy::DurabilityService (p. 1443)	**service_cleanup_delay** (p. 1445)
	<<***spec***>> *(p. 112)* Controls when the service is able to remove all information regarding a data instances.
dds::core::policy::DurabilityService (p. 1443)	**history_kind** (p. 1445)
	<<***spec***>> *(p. 112)* The kind of history to apply in recouping durable data.

dds::core::policy::DurabilityService (p. 1443)	**history_depth** (p. 1446)
	<<***spec***>> *(p. 112)* Setting to use for the **dds::core::policy::Durability::writer_depth** (p. 1439) when recouping durable data.
dds::core::policy::DurabilityService (p. 1443)	**max_samples** (p. 1446)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_instances** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
dds::core::policy::DurabilityService (p. 1443)	**max_samples_per_instance** (p. 1447)
	<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.
rti::core::CompressionSettings (p. 943)	**compression_ids** (p. 945)
	<<***extension***>> *(p. 116)* Mask that represents the compression algorithms enabled.
dds::core::policy::DataRepresentation (p. 1113)	**value** (p. 1116)
	<<***spec***>> *(p. 112)* Sequence of representation identifiers.
dds::core::policy::Presentation (p. 1942)	**access_scope** (p. 1947)
	<<***spec***>> *(p. 112)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
dds::core::policy::Presentation (p. 1942)	**coherent_access** (p. 1947)
	<<***spec***>> *(p. 112)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.
dds::core::policy::Presentation (p. 1942)	**ordered_access** (p. 1948)
	<<***spec***>> *(p. 112)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
dds::core::policy::LatencyBudget (p. 1631)	**duration** (p. 1633)
	<<***spec***>> *(p. 112)* Duration of the maximum acceptable delay.
dds::core::policy::Reliability (p. 2149)	**acknowledgment_kind** (p. 2154)
	<<***extension***>> *(p. 116)* Kind of reliable acknowledgment
dds::core::policy::Reliability (p. 2149)	**instance_state_consistency_kind** (p. 2155)

	<<***extension***>> *(p. 116)* Whether instance state consistency is enabled
dds::core::policy::Reliability (p. 2149)	**exact_reliability_kind_match** (p. 2155)
	<<***extension***>> *(p. 116)* Whether an exact reliability kind match is required for endpoint matching.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**kind** (p. 2565)
	<<***spec***>> *(p. 112)* Type consistency kind.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_sequence_bounds** (p. 2566)
	<<***spec***>> *(p. 112)* Controls whether sequence bounds are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_string_bounds** (p. 2567)
	<<***spec***>> *(p. 112)* Controls whether string bounds are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_member_names** (p. 2567)
	<<***spec***>> *(p. 112)* Controls whether member names are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**prevent_type_widening** (p. 2568)
	<<***spec***>> *(p. 112)* Controls whether type widening is allowed.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**force_type_validation** (p. 2568)
	<<***spec***>> *(p. 112)* Controls whether type information must be available in order to complete matching between a **dds::pub::DataWriter** (p. 1138) and a **dds::sub::↩ DataReader** (p. 976).
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_enum_literal_names** (p. 2569)
	<<***spec***>> *(p. 112)* Controls whether enumeration constant names are taken into consideration for type assignability.
rti::core::policy::TransportUnicast (p. 2556)	**value** (p. 2558)
	<<***extension***>> *(p. 116)* A sequence of unicast communication settings.
rti::core::policy::DataReaderProtocol (p. 1059)	**virtual_guid** (p. 1060)

	<<***extension***>> *(p. 116)* The virtual GUID (Global Unique Identifier).
rti::core::policy::DataReaderProtocol (p. 1059)	**expects_inline_qos** (p. 1062)
	[DEPRECATED] (p. **296)** <<***extension***>> *(p. 116)* Specifies whether this DataReader expects inline QoS with every sample.
rti::core::policy::DataReaderProtocol (p. 1059)	**disable_positive_acks** (p. 1062)
	<<***extension***>> *(p. 116)* Whether the reader sends positive acknowledgements to writers.
rti::core::policy::DataWriterProtocol (p. 1210)	**virtual_guid** (p. 1212)
	<<***extension***>> *(p. 116)* The virtual GUID (Global Unique Identifier).
rti::core::policy::DataWriterProtocol (p. 1210)	**disable_positive_acks** (p. 1214)
	<<***extension***>> *(p. 116)* Controls whether or not the writer expects positive acknowledgements from matching readers.
dds::core::policy::DataTag (p. 1132)	**tags** (p. 1137)
	<<***spec***>> *(p. 112)* Sequence of data tags.
rti::core::policy::EntityName (p. 1530)	**role_name** (p. 1532)
	<<***extension***>> *(p. 116)* The entity role name.
rti::core::policy::Service (p. 2350)	**kind** (p. 2354)
	<<***extension***>> *(p. 116)* The kind of service.
rti::core::policy::MultiChannel (p. 1749)	**channels** (p. 1751)
	<<***extension***>> *(p. 116)* A sequence of **rti::core::↩ ChannelSettings** (p. 921) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.
rti::core::policy::MultiChannel (p. 1749)	**filter_name** (p. 1752)
	<<***extension***>> *(p. 116)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.
rti::core::policy::TopicQueryDispatch (p. 2522)	**enable** (p. 2524)
	<<***extension***>> *(p. 116)* Allows this writer to dispatch TopicQueries

6.81 Matching

These QoS affect participant and endpoint matching.

These QoS affect participant and endpoint matching.

See the pages of each individual QoS for a desription of how the QoS must be set in order for two applications to be considered matching. Look for information about compatibility or request-offered (**RxO** (p. 514)) semantics.

Basic Settings

dds::core::policy::Durability (p. 1433)	**kind** (p. 1437)
	*<<**basic**>>* (p. *104*) *<<**spec**>>* (p. *112*) The kind of durability.
dds::core::policy::DataRepresentation (p. 1113)	**compression_settings** (p. 1117)
	*<<**basic**>>* (p. *104*) *<<**extension**>>* (p. *116*) Structure that contains the compression settings.
dds::core::policy::Deadline (p. 1255)	**period** (p. 1258)
	*<<**basic**>>* (p. *104*) *<<**spec**>>* (p. *112*) Duration of the deadline period.
dds::core::policy::Ownership (p. 1899)	**kind** (p. 1904)
	*<<**basic**>>* (p. *104*) *<<**spec**>>* (p. *112*) The kind of ownership
dds::core::policy::Liveliness (p. 1645)	**kind** (p. 1649)
	*<<**basic**>>* (p. *104*) *<<**spec**>>* (p. *112*) The kind of liveliness desired.
dds::core::policy::Liveliness (p. 1645)	**lease_duration** (p. 1649)
	*<<**basic**>>* (p. *104*) *<<**spec**>>* (p. *112*) The duration within which a **dds::pub::DataWriter** (p. 1138) must be asserted, or else it is assumed to be not alive.
dds::core::policy::Partition (p. 1923)	**name** (p. 1926)
	*<<**basic**>>* (p. *104*) *<<**spec**>>* (p. *112*) A list of partition names
dds::core::policy::Reliability (p. 2149)	**kind** (p. 2153)
	*<<**basic**>>* (p. *104*) *<<**spec**>>* (p. *112*) Kind of reliability.
dds::core::policy::DestinationOrder (p. 1258)	**kind** (p. 1261)
	*<<**basic**>>* (p. *104*) *<<**spec**>>* (p. *112*) Specifies the desired kind of destination order.

Advanced Settings

rti::core::CompressionSettings (p. 943)	**compression_ids** (p. 945)
	*<<**extension**>>* (p. *116*) Mask that represents the compression algorithms enabled.

dds::core::policy::DataRepresentation (p. 1113)	**value** (p. 1116)
	<<***spec***>> *(p. 112)* Sequence of representation identifiers.
dds::core::policy::Presentation (p. 1942)	**access_scope** (p. 1947)
	<<***spec***>> *(p. 112)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
dds::core::policy::Presentation (p. 1942)	**coherent_access** (p. 1947)
	<<***spec***>> *(p. 112)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.
dds::core::policy::Presentation (p. 1942)	**ordered_access** (p. 1948)
	<<***spec***>> *(p. 112)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
dds::core::policy::LatencyBudget (p. 1631)	**duration** (p. 1633)
	<<***spec***>> *(p. 112)* Duration of the maximum acceptable delay.
dds::core::policy::Reliability (p. 2149)	**acknowledgment_kind** (p. 2154)
	<<***extension***>> *(p. 116)* Kind of reliable acknowledgment
dds::core::policy::Reliability (p. 2149)	**instance_state_consistency_kind** (p. 2155)
	<<***extension***>> *(p. 116)* Whether instance state consistency is enabled
dds::core::policy::Reliability (p. 2149)	**exact_reliability_kind_match** (p. 2155)
	<<***extension***>> *(p. 116)* Whether an exact reliability kind match is required for endpoint matching.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**kind** (p. 2565)
	<<***spec***>> *(p. 112)* Type consistency kind.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_sequence_bounds** (p. 2566)
	<<***spec***>> *(p. 112)* Controls whether sequence bounds are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_string_bounds** (p. 2567)

	<<***spec***>> *(p. 112)* Controls whether string bounds are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_member_names** (p. 2567)
	<<***spec***>> *(p. 112)* Controls whether member names are taken into consideration for type assignability.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**prevent_type_widening** (p. 2568)
	<<***spec***>> *(p. 112)* Controls whether type widening is allowed.
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**force_type_validation** (p. 2568)
	<<***spec***>> *(p. 112)* Controls whether type information must be available in order to complete matching between a **dds::pub::DataWriter** (p. 1138) and a **dds::sub::DataReader** (p. 976).
dds::core::policy::TypeConsistencyEnforcement (p. 2563)	**ignore_enum_literal_names** (p. 2569)
	<<***spec***>> *(p. 112)* Controls whether enumeration constant names are taken into consideration for type assignability.

6.82 Deprecated QoS

Deprecated QoS.

Deprecated QoS.

This page lists the QoS that are deprecated. Deprecated QoS are still supported, but RTI will not support them in future releases.

dds::core::policy::ResourceLimits (p. 2203)	**instance_hash_buckets** (p. 2208)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Number of hash buckets for instances.
rti::core::policy::TypeSupport (p. 2574)	**plugin_data** (p. 2575)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Value to pass into the type plugin's de-/serialization function
rti::core::policy::TypeSupport (p. 2574)	**cdr_padding_kind** (p. 2576)
	[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Determines whether or not the padding bytes will be set to zero during CDR serialization.

rti::core::policy::AsynchronousPublisher (p. 832)	**disable_asynchronous_write** (p. 835)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Disable asynchronous publishing.
rti::core::policy::AsynchronousPublisher (p. 832)	**disable_asynchronous_batch** (p. 836)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Disable asynchronous batch flushing.
rti::core::policy::AsynchronousPublisher (p. 832)	**disable_topic_query_publication** (p. 838)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Disable topic query publication.
rti::core::policy::Discovery (p. 1265)	**enable_endpoint_discovery** (p. 1271)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Whether to automatically enable endpoint discovery for all the remote participants.
rti::core::policy::DataReaderResourceLimits (p. 1081)	**disable_fragmentation_support** (p. 1092)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) can receive fragmented samples.
rti::core::policy::DataWriterResourceLimits (p. 1235)	**initial_concurrent_blocking_threads** (p. 1238)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The initial number of threads that are allowed to concurrently block on write call on the same **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**max_concurrent_blocking_threads** (p. 1238)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum number of threads that are allowed to concurrently block on write call on the same **dds::pub::DataWriter** (p. 1138).
rti::core::policy::DataWriterResourceLimits (p. 1235)	**autoregister_instances** (p. 1242)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Whether or not to automatically register new instances
rti::core::policy::DataReaderProtocol (p. 1059)	**expects_inline_qos** (p. 1062)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Specifies whether this DataReader expects inline QoS with every sample.
rti::core::policy::SystemResourceLimits (p. 2443)	**max_objects_per_thread** (p. 2445)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum number of objects that can be stored per thread for a DomainParticipantFactory.

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_writer_hash_buckets** (p. 1406)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Hash_Buckets settings applied to local DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_reader_hash_buckets** (p. 1407)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_publisher_hash_buckets** (p. 1407)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local Publisher.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_subscriber_hash_buckets** (p. 1408)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local Subscriber.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**local_topic_hash_buckets** (p. 1408)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local Topic.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_writer_hash_buckets** (p. 1409)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote DataWriters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_reader_hash_buckets** (p. 1409)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote DataReaders.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_participant_hash_buckets** (p. 1410)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote DomainParticipants.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_writer_reader_pair_hash_buckets** (p. 1410)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for matching local writer and remote/local reader pairs.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**matching_reader_writer_pair_hash_buckets** (p. 1411)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for matching local reader and remote/local writer pairs.

rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**ignored_entity_hash_buckets** (p. 1411)
	[DEPRECATED] (p. **296**) <<***extension***>> (p. *116*) Number of hash buckets for ignored entities.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filtered_topic_hash_buckets** (p. 1412)
	[DEPRECATED] (p. **296**) <<***extension***>> (p. *116*) Number of hash buckets for content filtered topics.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**content_filter_hash_buckets** (p. 1412)
	[DEPRECATED] (p. **296**) <<***extension***>> (p. *116*) Number of hash buckets for content filters.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**flow_controller_hash_buckets** (p. 1413)
	[DEPRECATED] (p. **296**) <<***extension***>> (p. *116*) Number of hash buckets for flow controllers.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**type_code_max_serialized_length** (p. 1418)
	[DEPRECATED] (p. **296**) <<***extension***>> (p. *116*) Maximum size of serialized string for type code.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**ignored_entity_replacement_kind** (p. 1429)
	[DEPRECATED] (p. **296**) <<***extension***>> (p. *116*) Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **rti::core::policy::DomainParticipant↩ ResourceLimits::ignored_entity_allocation** (p. 1403) are reached.
rti::core::policy::DomainParticipantResourceLimits (p. 1391)	**remote_topic_query_hash_buckets** (p. 1430)
	[DEPRECATED] (p. **296**) <<***extension***>> (p. *116*) Number of hash buckets for remote TopicQueries.
rti::core::policy::Database (p. 969)	**shutdown_timeout** (p. 972)
	[DEPRECATED] (p. **296**) <<***extension***>> (p. *116*) The maximum wait time during a shutdown.
rti::core::policy::Database (p. 969)	**cleanup_period** (p. 972)
	[DEPRECATED] (p. **296**) <<***extension***>> (p. *116*) The period at which the database thread wakes up to remove deleted records.
rti::core::policy::Database (p. 969)	**shutdown_cleanup_period** (p. 973)

	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The clean-up period used during database shut-down.
rti::core::policy::Database (p. 969)	**max_skiplist_level** (p. 974)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum level of the skiplist.
rti::core::policy::ReceiverPool (p. 2144)	**buffer_alignment** (p. 2148)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The receive buffer alignment.
rti::core::policy::DiscoveryConfig (p. 1271)	**remote_participant_purge_kind** (p. 1281)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
rti::core::policy::DiscoveryConfig (p. 1271)	**max_liveliness_loss_detection_period** (p. 1281)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.
rti::core::policy::DiscoveryConfig (p. 1271)	**ignore_default_domain_announcements** (p. 1301)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.
rti::core::policy::DiscoveryConfig (p. 1271)	**participant_message_reader_reliability_kind** (p. 1295)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Reliability policy for a built-in participant message reader.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 2247)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Enables dynamic adjustment of sample keep duration in response to congestion.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_decrease_sample_keep_duration_factor** (p. 2248)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Controls rate of contraction of dynamic sample keep duration.
rti::core::policy::RtpsReliableWriterProtocol (p. 2229)	**disable_positive_acks_increase_sample_keep_duration_factor** (p. 2248)
	[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Controls rate of growth of dynamic sample keep duration.

rti::core::policy::BuiltinTopicReaderResourceLimits (p. 906)	**disable_fragmentation_support** (p. 913)
	[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Determines whether the built-in topic **dds::sub::Data↩ Reader** (p. 976) can receive fragmented samples.

6.83 System Properties

System Properties.

System Properties.

RTI Connext uses the **rti::core::policy::Property** (p. 1969) of a DomainParticipant to maintain a set of properties that provide system information such as hostname.

Unless the default **dds::domain::qos::DomainParticipantQos** (p. 1384) value is overwritten, the system properties are automatically set in the **dds::domain::qos::DomainParticipantQos** (p. 1384) obtained by calling the method **dds::domain::DomainParticipant::default_participant_qos()** (p. 1341) or using the constant PARTICIPANT_QOS↩ _DEFAULT.

System properties are also automatically set in the **dds::domain::qos::DomainParticipantQos** (p. 1384) loaded from an XML QoS profile unless you disable property inheritance using the attribute **inherit** in the XML tag <**property**>.

By default, the system properties are propagated to other DomainParticipants in the system and can be accessed through **dds::topic::ParticipantBuiltinTopicData::property** (p. 1910).

You can disable the propagation of individual properties by setting the flag Property_t::propagate to false or by removing the property using the method PropertyQosPolicyHelper::remove_property.

The number of system properties set on the **dds::domain::qos::DomainParticipantQos** (p. 1384) is platform specific.

6.83.1 System Properties List

The following table lists the supported system properties. For more information, see `System Properties`, in the Core Libraries User's Manual.

Property Name	Description
dds.sys_info.hostname	Name of the host where the process is running
dds.sys_info.process_id	Process ID
dds.sys_info.username	Name of the user running the process
dds.sys_info.execution_timestamp	Time when the execution started
dds.sys_info.creation_timestamp	Time when the executable was created
dds.sys_info.executable_filepath	Name and full path of the executable
dds.sys_info.target	Architecture for which the library was compiled

6.83.2 System Resource Consideration

System properties are affected by the resource limits **rti::core::policy::DomainParticipantResourceLimits↩ ::participant_property_list_max_length** (p. 1424) and **rti::core::policy::DomainParticipantResourceLimits↩ ::participant_property_string_max_length** (p. 1425).

6.84 Configuring QoS Profiles with XML

APIs related to XML QoS Profiles.

Classes

- class **dds::core::QosProvider**

  *<<**reference-type**>> (p. 351) The **QosProvider** (p. 2025) class provides a way for a user to control and access the XML QoS profiles that are loaded by RTI Connext*
- class **rti::core::QosProviderParams**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Configure options that control the way that XML documents containing QoS profiles are loaded by a **dds::core::QosProvider** (p. 2025).*

6.84.1 Detailed Description

APIs related to XML QoS Profiles.

6.84.2 Loading QoS Profiles from XML Resources

A 'QoS profile' is a group of QoS settings, specified in XML format. By using QoS profiles, you can change QoS settings without recompiling the application.

The Qos profiles are loaded the first time any of the following operations are called:

- Whenever a **dds::domain::DomainParticipant** (p. 1326) is created
- **dds::core::QosProvider::participant_qos(const std::string& id)** (p. 2031)
- **dds::domain::DomainParticipant::default_participant_qos()** (p. 1341)
- **dds::core::QosProvider::default_library(const std::string& library_name)** (p. 2041)
- **dds::core::QosProvider::default_profile(const std::string& profile_name)** (p. 2041)
- **dds::core::QosProvider::participant_qos(const std::string& id)** (p. 2031)
- **dds::core::QosProvider::topic_qos(const std::string& id)** (p. 2032)

- **dds::core::QosProvider::topic_qos_w_topic_name(const std::string& topic_name) const** (p. 2038)
- **dds::core::QosProvider::topic_qos_w_topic_name(const std::string& id, const std::string& topic_name) const** (p. 2038)
- **dds::core::QosProvider::publisher_qos(const std::string& id)** (p. 2034)
- **dds::core::QosProvider::subscriber_qos(const std::string& id)** (p. 2033)
- **dds::core::QosProvider::datawriter_qos(const std::string& id)** (p. 2035)
- **dds::core::QosProvider::datawriter_qos_w_topic_name(const std::string& topic_name) const** (p. 2040)
- **dds::core::QosProvider::datawriter_qos_w_topic_name(const std::string& id, const std::string& topic_name) const** (p. 2040)
- **dds::core::QosProvider::datareader_qos(const std::string& id)** (p. 2033)
- **dds::core::QosProvider::datareader_qos_w_topic_name(const std::string& topic_name) const** (p. 2039)
- **dds::core::QosProvider::topic_qos_w_topic_name(const std::string& id, const std::string& topic_name) const** (p. 2038)
- **dds::core::QosProvider::qos_profile_libraries()** (p. 2043)
- **dds::core::QosProvider::qos_profiles()** (p. 2043)
- **dds::core::QosProvider::load_profiles()** (p. 2045)

The QoS profiles are reloaded replacing previously loaded profiles when the following operations are called:

- **dds::domain::DomainParticipant::participant_factory_qos(const dds::domain::qos::DomainParticipantFactoryQos& qos)** (p. 1340)
- **dds::core::QosProvider::reload_profiles()** (p. 2045)

The **dds::core::QosProvider::unload_profiles()** (p. 2045) operation will free the resources associated with the XML QoS profiles.

There are five ways to configure the XML resources (listed by load order):

- The file NDDS_QOS_PROFILES.xml in $NDDSHOME/resource/xml is loaded if it exists and **rti::core::QosProviderParams::ignore_resource_profile** (p. 2052) in **rti::core::QosProviderParams** (p. 2049) is set to false (first to be loaded). An example file, `NDDS_QOS_PROFILES.example.xml`, is available for reference.

- The URL groups separated by semicolons referenced by the enviroment variable NDDS_QOS_PROFILES are loaded if they exist and **rti::core::QosProviderParams::ignore_environment_profile** (p. 2052) in **rti::core::QosProviderParams** (p. 2049) is set to false.

- The file USER_QOS_PROFILES.xml in the working directory will be loaded if it exists and **rti::core::QosProviderParams::ignore_user_profile** (p. 2051) in **rti::core::QosProviderParams** (p. 2049) is set to false.

- The URL groups referenced by **rti::core::QosProviderParams::url_profile** (p. 2050) in **rti::core::QosProviderParams** (p. 2049) will be loaded if specified.

- The sequence of XML strings referenced by **rti::core::QosProviderParams::string_profile** (p. 2050) will be loaded if specified (last to be loaded).

The above methods can be combined together.

6.84.3 URL

The location of the XML resources (only files and strings are supported) is specified using a URL (Uniform Resource Locator) format. For example:

File Specification: `file:///usr/local/default_dds.xml`

String Specification: str://"<dds><qos_library> . . . lt;/qos_library>&lt/dds>"

If the URL schema name is omitted, RTI Connext will assume a file name. For example:

File Specification: /usr/local/default_dds.xml

6.84.3.1 URL groups

To provide redundancy and fault tolerance, you can specify multiple locations for a single XML document via URL groups. The syntax of a URL group is as follows:

[URL1 | URL2 | URL2 | . . .| URLn]

For example:

[`file:///usr/local/default_dds.xml` | `file:///usr/local/alternative_default_dds.xml`]

Only one of the elements in the group will be loaded by RTI Connext, starting from the left.

Brackets are not required for groups with a single URL.

6.84.3.2 NDDS_QOS_PROFILES environment variable

The environment variable NDDS_QOS_PROFILES contains a list of URL groups separated by ';'

The URL groups referenced by the environment variable are loaded if they exist and **rti::core::QosProviderParams::ignore_environment_profile** (p. 2052) is set to false

6.84.3.3 Built-In QoS Profiles

There are also a number of built-in QoS profiles that can be used without having to load any configurations from outside XML resources. For more information on these built-in profiles see **Built-in Qos Profiles** (p. 462).

For more information on XML Configuration, refer to the `User's Manual`.

6.85 Publication Example

A data publication example.

A data publication example.

6.85.1 A typical publication example

Prep

- Create user data types using rtiddsgen. See the `Code Generator User's Manual`.

Set up

- **Set up participant** (p. 307)
- **Set up publisher** (p. 310)
- **Register user data type(s)** (p. 308)
- **Set up topic(s)** (p. 308)
- **Set up data writer(s)** (p. 311)

Adjust the desired quality of service (QoS)

- **Adjust QoS on entities as necessary** (p. 325)

Send data

- **Send data** (p. 312)

Tear down

- **Tear down all entities** (p. 328)

6.86 Subscription Example

A data subscription example.

A data subscription example.

6.86.1 A typical subscription example

Prep

- Create user data types using rtiddsgen. See the `Code Generator User's Manual`.

Set up

- **Set up participant** (p. 307)
- **Set up subscriber** (p. 315)
- **Register user data type(s)** (p. 308)
- **Set up topic(s)** (p. 308)
- **Set up data reader(s)** (p. 316)
- **Set up DataReader** (p. 317) to receive data

Adjust the desired quality of service (QoS)

- **Adjust QoS on entities as necessary** (p. 325)

Receive data

- **Read all samples** (p. 318) or **select which samples** (p. 319) to read from the DataReader queue

Tear down

- **Tear down all entities** (p. 328)

6.87 Participant Use Cases

Working with domain participants.

Working with domain participants.

6.87.1 Turning off auto-enable of newly created participant(s)

- Change the value of the **ENTITY_FACTORY** (p. 532) for the DomainParticipantFactory

  The following #includes are needed for the examples on this page
  ```
  #include <dds/domain/ddsdomain.hpp>
  ```

6.87.2 Setting up a DomainParticipant

- How to create DomainParticipants with each of the available constructors

```
// Creating a DomainParticipants on domain 0, with default Qos, and no
// listener
dds::domain::DomainParticipant participant1(0) ;
// Creating a DomainParticipants with a non-default QoS values
dds::domain::qos::DomainParticipantQos qos;
 // Set the initial peers.
 // The peers stored in initial_peers are merely potential peers--there is
 // no requirement that the peer DomainParticipants are actually up and
 // running or even will eventually exist. The Connext discovery process
 // will try to contact all potential peer participants in the list
 // periodically using unicast transports.
 const char * initial_peers_array[] = {
   "host1",
   "10.10.30.192",
   "1@localhost",
   "2@host2",
   "my://", /* all unicast addrs on transport plugins with alias "my" */
   "2@shmem://", /* shared memory */
   "FF00:ABCD::0",
   "1@FF00:0:1234::0",
   "225.1.2.3",
   "3@225.1.0.55",
   "FAA0::0#0/0/R"};
const dds::core::StringSeq initial_peers_list(initial_peers_array, initial_peers_array+11);
// Add the initial peers list to the DiscoveryQosPolicy while modifying
// the participant's qos inline
qos << rti::core::policy::Discovery().initial_peers(initial_peers_list);
// Creating a DomainParticipants on domain 1 with modified QoS
dds::domain::DomainParticipant participant2(1, qos);
// Creating a DomainParticipants on domain 2, with modified QoS, and a
// listener
auto listener = std::make_shared<ExampleParticipantListener>();
dds::domain::DomainParticipant participant3(2, qos, listener);
// Creating a DomainParticipants on domain 3, with modified QoS, a listener
// and a data_available() status mask
dds::domain::DomainParticipant participant4(
        3,
        qos,
        listener,
        dds::core::status::StatusMask::data_available());
```

- Setting the DomainParticipantFactoryQos so that participants are created disabled (optional)

```
dds::domain::qos::DomainParticipantFactoryQos qos;
qos << dds::core::policy::EntityFactory::ManuallyEnable();
dds::domain::DomainParticipant::participant_factory_qos(qos);
```

6.87.3 Looking up DomainParticipants

- Lookup DomainParticipants by domain id

```
// Create and retain two DomainParticipants on domain 0 and one on domain 1
create_and_retain_participant(0);
create_and_retain_participant(0);
create_and_retain_participant(1);
// We don't have any references to the created DomainParticipants but we
// can look them up using the domain id they were created on
// If there is more than one participant on the same domain then this
// operation will return only one of them. It is unspecified which one it
// will return
dds::domain::DomainParticipant participant1 = dds::domain::find(0);
dds::domain::DomainParticipant participant2 = dds::domain::find(1);
```

6.87.4 Tearing down a Participant

- See **Tearing Down An Entity** (p. 328) for eamples of the various ways to teardown a Participant

6.87.5 Finalizing the factory

- Finalizing the factory

```
// The DomainParticipantFactory is a singleton that the C++ API uses
// implicitly to create participants. RTI Connext provides
// dds::domain::DomainParticipant::finalize_participant_factory()
// for users who want to release memory used by the participant factory.
dds::domain::DomainParticipant::finalize_participant_factory();
```

6.88 Topic Use Cases

The following #includes are needed for the examples on this page

```
#include <dds/domain/ddsdomain.hpp>
#include <dds/core/ddscore.hpp>
#include <dds/topic/ddstopic.hpp>
#include "Foo.hpp"
```

6.88.1 Registering a user data type

- Types are implicitly registered during Topic construction, see **Setting up a Topic** (p. 308)

6.88.2 Setting up a Topic

- **Set up a DomainParticipant** (p. 307)
- Create a Topic with each of the available constructors

```
// Create the participant
dds::domain::DomainParticipant participant(0);
// Creating a topic with user-defined type Foo and topic name MyTopic1,
// with default Qos and no listener
dds::topic::Topic<Foo> topic1(participant, "MyTopic1");
// Creating a topic with user-defined type Foo, topic name MyTopic2,
// type name MyTypeName2
dds::topic::Topic<Foo> topic2(participant, "MyTopic2", "MyTypeName2");
// Creating a topic with user-defined type Foo, topic name MyTopic3, and
// modified QoS
dds::topic::qos::TopicQos qos;
qos << dds::core::policy::Durability::TransientLocal();
dds::topic::Topic<Foo> topic3(participant, "MyTopic3", qos);
// Creating a topic with user-defined type Foo, topic name MyTopic4, type
// name MyTypeName4 and modified QoS
dds::topic::Topic<Foo> topic4(participant, "MyTopic4", "MyTypeName4", qos);
// Creating a topic with user-defined type Foo, topic name MyTopic5,
// modified QoS, and a listener
auto listener = std::make_shared<ExampleTopicListener>();
dds::topic::Topic<Foo> topic5(participant, "MyTopic5", qos, listener);
// Creating a topic with user-defined type Foo, topic name MyTopic6, type
// name MyTypeName6, modified QoS, and a listener
dds::topic::Topic<Foo> topic6(
   participant, "MyTopic6", "MyTypeName6", qos, listener);
// Creating a topic with user-defined type Foo, topic name MyTopic7,
// modified QoS, a non-NULL listener, and a inconsistent_topic status mask
dds::topic::Topic<Foo> topic7(
   participant, "MyTopic7", qos, listener,
   dds::core::status::StatusMask::inconsistent_topic());
// Creating a topic with user-defined type Foo, topic name MyTopic8, type
// name MyTypeName8, modified QoS, a listener, and
// a inconsistent_topic status mask
dds::topic::Topic<Foo> topic8(
   participant, "MyTopic8", "MyTypeName8", qos, listener,
   dds::core::status::StatusMask::inconsistent_topic());
```

6.88.3 Discovering Topics

- Lookup local Topics by their topic name

```
void howto_lookup_topics()
{
    dds::domain::DomainParticipant participant(0);
    // Create two Topics and retain them
    create_and_retain_topic(participant, "Topic1");
    create_and_retain_topic(participant, "Topic2");
    // We don't have any references to the created Topics but we can look
    // them up using topic name that they were created with
    dds::topic::Topic<Foo> topic1 =
        dds::topic::find<dds::topic::Topic<Foo> >(participant, "Topic1");
    dds::topic::Topic<Foo> topic2 =
        dds::topic::find<dds::topic::Topic<Foo> >(participant, "Topic2");
    // We retained the topics so we must close them now
    topic1.close();
    topic2.close();
}
void create_and_retain_topic(
   dds::domain::DomainParticipant &participant, const std::string& topic_name)
{
    dds::topic::Topic<Foo> topic(participant, topic_name);
    topic.retain();
}
```

- The next two examples assume the following setup

```
    // Create two DomainParticipants on the same domain
    dds::domain::DomainParticipant participant1(1);
    dds::domain::DomainParticipant participant2(1);

    // Create some Topics for both of the Participants
    dds::topic::Topic<Foo> topic1(participant1, "Topic1");
    dds::topic::Topic<Foo> topic2(participant1, "Topic2");
    dds::topic::Topic<Foo> topic3(participant2, "Topic3");
```

- Discover all Topics in a domain

```
    // Get a list of the InstanceHandles of all of the topics in the
    // domain. topic_count will = 3
    dds::core::InstanceHandleSeq handles1;
    int topic_count = dds::topic::discover_any_topic(
       participant1, std::back_inserter(handles1));
    // discover_any_topic also takes a forward iterator and a max number of
    // topics to discover. topic_count will = 2. There is no guarantee
    // as to which two topics' handles will be retrieved
    dds::core::InstanceHandleSeq handles2(10, dds::core::InstanceHandle::nil());
    topic_count = dds::topic::discover_any_topic(participant2, handles2.begin(), 2);
```

- Retrieve the TopicBuiltinTopicData for a Topic

```
    std::vector<dds::topic::TopicBuiltinTopicData> topic_data1;
    // Get the TopicBuiltinTopicData for all of the participant's local topics
    // topic_count will = 2 because participant1 has 2 associated Topics
    topic_count = dds::topic::discover_topic_data(participant1, std::back_inserter(topic_data1));
    std::vector<dds::topic::TopicBuiltinTopicData> topic_data2(10);
    // discover_any_topic also takes a forward iterator and a max number of
    // topics to get the data for. Because we're using participant2,
    // topic_count will = 1
    topic_count = dds::topic::discover_topic_data(participant2, topic_data2.begin());
    // You can also retrieve data about a specific Topic or Topics with their
    // InstanceHandles. Because discover_topic_data will only retrieve data
    // about local topics, using an InstanceHandle that refers to a remote
    // Topic will cause an UnsupportedError to be thrown in either of the
    // following cases
    // Retrieve the TopicBuiltinTopicData for topic3
    dds::topic::TopicBuiltinTopicData topic_data =
        dds::topic::discover_topic_data(participant2, topic3.instance_handle());
    // Pass in a container with multiple handles and retrieve the
```

```
// TopicBuiltinTopicData for each of the Topics associated with each of the
// InstanceHandles
std::vector<dds::topic::TopicBuiltinTopicData> topic_data3(10);
dds::core::InstanceHandleSeq local_handles;
local_handles.push_back(topic1.instance_handle());
local_handles.push_back(topic2.instance_handle());
// topic_count will equal 2
topic_count = dds::topic::discover_topic_data(
   participant1, topic_data3.begin(), local_handles);
```

6.88.4 Tearing down a topic

- See **Tearing Down An Entity** (p. 328) for eamples of the various ways to teardown a Topic

6.89 Publisher Use Cases

The following #includes are needed for the examples on this page

```
#include <dds/pub/ddspub.hpp>
#include <dds/domain/ddsdomain.hpp>
#include <dds/core/ddscore.hpp>
```

6.89.1 Setting up a publisher

- **Set up participant** (p. 307)
- Create a Publisher with each of the available constructors

```
// Create the participant
dds::domain::DomainParticipant participant(0);
// When not explicitly provided to the constructor, the
// created publisher's QoS is default, has a NULL listener, and a status mask
// dds::core::status::StatusMask::all()
// Creating a publisher
dds::pub::Publisher publisher1(participant);
// Creating a publisher with modified QoS
dds::pub::qos::PublisherQos qos;
qos << rti::core::policy::EntityName("PublisherName");
dds::pub::Publisher publisher2(participant, qos);
// Creating a publisher with modified QoS and a listener
auto listener = std::make_shared<ExamplePublisherListener>();
dds::pub::Publisher publisher3(participant, qos, listener);
// Creating a publisher with modified QoS, a listener, and a
// sample_rejected() status mask
dds::pub::Publisher publisher4(
        participant,
        qos,
        listener,
        dds::core::status::StatusMask::offered_deadline_missed());
```

6.89.2 Looking up Publishers

- Lookup Publishers

```
void howto_lookup_publishers()
{
    dds::domain::DomainParticipant participant(0);
    // Create three Publishers and retain them
    create_and_retain_publisher(participant);
    create_and_retain_publisher(participant);
    create_and_retain_publisher(participant);
    // We don't have any references to the created Publishers but we can look
    // them up using the participant that they were created with
```

```
    std::vector<dds::pub::Publisher> publishers1;
    // publisher_count will equal 3 and publishers1 will hold a reference to
    // each of the three publishers that were created
    int publisher_count =
        rti::pub::find_publishers(participant, std::back_inserter(publishers1));
    // rti::pub::publishers also takes a forward iterator and a max number of
    // publishers to look up
    std::vector<dds::pub::Publisher> publishers2(2, dds::pub::Publisher(dds::core::null));
    // publisher_count will equal 2 because we're only asking for 2
    publisher_count =
        rti::pub::find_publishers(participant, publishers2.begin(), 2);
    std::cout << "Publisher count: " << publisher_count << std::endl;
    // We retained the publishers so we must close them now
    for (std::vector<dds::pub::Publisher>::iterator it = publishers1.begin();
          it != publishers1.end();
          it++) {
        (*it).close();
    }
}
void create_and_retain_publisher(dds::domain::DomainParticipant &participant)
{
    dds::pub::Publisher publisher(participant);
    publisher.retain();
}
```

6.89.3 Tearing down a publisher

- See **Tearing Down An Entity** (p. 328) for eamples of the various ways to teardown a Publisher

6.90 DataWriter Use Cases

The following #includes are needed for the examples on this page

```
#include <dds/pub/ddspub.hpp>
#include <dds/domain/ddsdomain.hpp>
#include <dds/core/ddscore.hpp>
#include <dds/topic/ddstopic.hpp>
#include <dds/sub/ddssub.hpp>
#include "Foo.hpp"
```

6.90.1 Setting up a DataWriter

- **Set up a Publisher** (p. 310)
- **Set up a Topic** (p. 308)
- Create a DataWriter of user-defined type **Foo** (p. 1588) with each of the available constructors

```
    // Create the Participant
    dds::domain::DomainParticipant participant(0);
    // Create a Topic
    dds::topic::Topic<Foo> topic(participant, "MyTopicName");
    // When not explicitly provided to the constructor, the
    // created DataWriter's QoS is default, has a NULL listener, and a status mask
    // dds::core::status::StatusMask::all()
    // Creating a DataWriter, creating the subscriber inline
    dds::pub::DataWriter<Foo> writer1(dds::pub::Publisher(participant), topic);
    // Create a Publisher
    dds::pub::Publisher publisher(participant);
    // Create a DataWriter with modified QoS
    dds::pub::qos::DataWriterQos writer_qos =
            dds::core::QosProvider::Default().datawriter_qos();
    writer_qos << dds::core::policy::Liveliness::ManualByParticipant();
    dds::pub::DataWriter<Foo> writer2(publisher, topic, writer_qos);
    // Create a DataWriter with the Qos of a topic
    writer_qos = topic.qos(); // Copy the policies in TopicQos into a DataWriterQos
```

```
dds::pub::DataWriter<Foo> writer3(publisher, topic, writer_qos);
// Create a DataWriter with modified QoS and a listener
auto listener = std::make_shared<ExampleDataWriterListener<Foo»();
dds::pub::DataWriter<Foo> writer4(
   publisher,
   topic,
   writer_qos,
   listener);
// Create a DataWriter with modified QoS, a listener, and a liveliness_lost()
// status mask
dds::pub::DataWriter<Foo> writer5(
   publisher,
   topic,
   writer_qos,
   listener,
   dds::core::status::StatusMask::liveliness_lost());
```

6.90.2 Managing instances

- The snippets in this section assume the following setup
```
dds::domain::DomainParticipant participant(0);
dds::topic::Topic<Foo> topic(participant, "MyTopic");
dds::pub::DataWriter<Foo> writer(dds::pub::Publisher(participant), topic);
Foo sample;
sample.x(100);
```

- Getting an instance "key" value of user data type **Foo** (p. 1588)
```
// Instead of a sample of type Foo, key_value() could also take a
// TopicInstance<Foo> object. key_value only populates the key field in the
// sample
Foo retrieved_sample_key;
writer.key_value(retrieved_sample_key, handle);
```

- Registering an instance of type **Foo** (p. 1588)
```
dds::core::InstanceHandle handle = writer.register_instance(sample);
```

- Unregistering an instance of type **Foo** (p. 1588)
```
writer.unregister_instance(handle);
```

- Disposing of an instance of type **Foo** (p. 1588)
```
writer.dispose_instance(handle);
```

6.90.3 Sending data

- **Set up a DataWriter** (p. 311)
- **Register an instance** (p. 312)

- Write instances of type **Foo** (p. 1588) (see also **Working with IDL types** (p. 606))

```
Foo data;
dds::core::InstanceHandle handle =
    dds::core::InstanceHandle::nil();
dds::core::Time timestamp(123);
// Write a sample
writer.write(data);
// You can also write a sample with a timestamp that will be used as the
// source timestamp for this sample
writer.write(data, timestamp);
// You can also write a sample with an InstanceHandle
writer.write(data, handle);
// The above call to write is equivalent to the following
writer.write(dds::topic::TopicInstance<Foo>(handle, data));
// Or a combination of handle and timestamp
writer.write(data, handle, timestamp);
// The above call to write is equivalent to the following
writer.write(dds::topic::TopicInstance<Foo>(handle, data), timestamp);
// Create a vector of three default-constructed samples
std::vector<Foo> samples(3, Foo());
// You can also write samples that are stored in a container by
// passing in an iterator range to write()
writer.write(samples.begin(), samples.end());
// You can add a timestamp to the above call to write
writer.write(samples.begin(), samples.end(), timestamp);
// Create a vector of three instance handles
std::vector<dds::core::InstanceHandle> handles(
   3, dds::core::InstanceHandle::nil());
// Write 3 samples along with their handles
writer.write(samples.begin(), samples.end(),
             handles.begin(), handles.end());
// You can add a timestamp to the above call to write
writer.write(samples.begin(), samples.end(),
             handles.begin(), handles.end(),
             timestamp);
// You can also use operator << to write samples
writer << Foo(1, 2)
       << std::make_pair(Foo(3, 4), timestamp) // add a timestamp
       << std::make_pair(Foo(5, 6), handle); // add an instance handle
```

6.90.4 DataWriter Listeners

- Implementing a DataWriterListener class

```
// To create a DataWriterListener, inherit from the NoOpDataWriterListener class
// and then override any of the listener callback functions
template <typename T>
class ExampleDataWriterListener : public dds::pub::NoOpDataWriterListener<T>
{
public:
    ExampleDataWriterListener() {}
public:
    void on_publication_matched(
       dds::pub::DataWriter<T>&,
       const dds::core::status::PublicationMatchedStatus &status)
    {
        std::cout << "on_publication_matched callback" << std::endl;
        // Access information from the status
        std::cout << "total_count = " << status.total_count() << std::endl;
        std::cout << "total_count_change = " << status.total_count_change() << std::endl;
        std::cout << "current_count = " << status.current_count() << std::endl;
        std::cout << "current_count_change = " << status.current_count_change() << std::endl;
        dds::core::InstanceHandle handle = status.last_subscription_handle();
        // Extension
        std::cout << "current_count_peak = "
                  << status.extensions().current_count_peak() << std::endl;
    }
    void on_offered_incompatible_qos(
       dds::pub::DataWriter<T>&,
       const dds::core::status::OfferedIncompatibleQosStatus &status)
    {
        std::cout << "on_offered_incompatible_qos callback" << std::endl;
        // Access information from the status
        std::cout << "total_count = " << status.total_count() << std::endl;
        std::cout << "total_count_change = " << status.total_count_change() << std::endl;
        std::cout << "last_policy_id = " << status.last_policy_id() << std::endl;
        dds::core::policy::QosPolicyCountSeq qos_seq = status.policies();
```

```
            if (qos_seq.size() > 0) {
                std::cout « "policy_id of one first incompatible Qos policy = "
                    « qos_seq[0].policy_id() « std::endl;
            }
        }
        void on_offered_deadline_missed(
           dds::pub::DataWriter<T>&,
           const dds::core::status::OfferedDeadlineMissedStatus &status)
        {
            std::cout « "on_offered_deadline_missed callback" « std::endl;
            // Access information from the status
            std::cout « "total_count = " « status.total_count() « std::endl;
            std::cout « "total_count_change = " « status.total_count_change() « std::endl;
            dds::core::InstanceHandle handle = status.last_instance_handle();
        }
        void on_liveliness_lost(
           dds::pub::DataWriter<T>&,
           const dds::core::status::LivelinessLostStatus &status)
        {
            std::cout « "on_liveliness_lost callback" « std::endl;
            // Access information from the status
            std::cout « "total_count = " « status.total_count() « std::endl;
            std::cout « "total_count_change = " « status.total_count_change() « std::endl;
        }
    };
```

- Setting a listener on a DataWriter

```
    dds::domain::DomainParticipant participant(0);
    dds::topic::Topic<Foo> topic(participant, "Topic1");
    auto listener = std::make_shared<ExampleDataWriterListener<Foo»();
    // Create a DataWriter with a listener and default
    // dds::core::status::StatusMask::all()
    dds::pub::DataWriter<Foo> writer1(
            dds::pub::Publisher(participant),
            topic,
            dds::pub::qos::DataWriterQos(),
            listener);
    // Create a DataWriter without a listener
    dds::pub::DataWriter<Foo> writer2(dds::pub::Publisher(participant), topic);
    // Set the listener after creation, specifying a status mask
    writer2.set_listener(
            listener,
            dds::core::status::StatusMask::offered_incompatible_qos());
```

6.90.5 Looking up DataWriters

- Lookup DataWriters by topic name

```
    dds::domain::DomainParticipant participant(0);
    dds::pub::Publisher publisher(participant);
    dds::topic::Topic<Foo> topic1(participant, "Topic1");
    dds::topic::Topic<Foo> topic2(participant, "Topic2");
    // Create three DataWriters for topic1 and one for topic2 and retain them
    create_and_retain_writer(publisher, topic1);
    create_and_retain_writer(publisher, topic1);
    create_and_retain_writer(publisher, topic2);
    // We don't have any references to the created DataWriters but we can look
    // them up using the publisher and topic name that they were created with
    std::vector<dds::pub::DataWriter<Foo> > writers;
    // writer_count will equal 1. If more than one DataWriter belongs to a
    // publisher with the same topic name then find may return any one of them
    int writer_count =
        dds::pub::find<dds::pub::DataWriter<Foo> >(
           publisher,
           "Topic1",
           std::back_inserter(writers));
    writer_count =
        dds::pub::find<dds::pub::DataWriter<Foo> >(
           publisher,
           "Topic2",
           std::back_inserter(writers));
```

For more examples, see **Looking up DataReaders** (p. 322)

6.90.6 Getting Matched Subscriptions

- Lookup a DataWriter's matched subscriptions

```
dds::domain::DomainParticipant participant(0);
dds::topic::Topic<Foo> topic(participant, "Topic1");
dds::pub::DataWriter<Foo> writer(dds::pub::Publisher(participant), topic);
// Create two readers that will match with the above writer
dds::sub::DataReader<Foo> reader1(dds::sub::Subscriber(participant), topic);
dds::sub::DataReader<Foo> reader2(dds::sub::Subscriber(participant), topic);
// Get a list of the InstanceHandles corresponding to the DataWriter's
// matched subscriptions.
dds::core::InstanceHandleSeq subscription_handles1 =
    dds::pub::matched_subscriptions(writer);
// matched_subscriptions also takes in an iterator range and will return
// an InstanceHandle iterator to the end of the InstanceHandleSeq
dds::core::InstanceHandleSeq subscription_handles2(2, dds::core::null);
dds::core::InstanceHandleSeq::iterator last =
    dds::pub::matched_subscriptions(
       writer,
       subscription_handles2.begin(),
       subscription_handles2.end());
// now last == subscription_handles2.end()
```

- Use the InstanceHandles obtained from **matched_subscriptions()** (p. 1187) to get the SubscriptionBuiltinTopic↩ Data for the subscriptions

```
dds::topic::SubscriptionBuiltinTopicData subscription_data =
    dds::pub::matched_subscription_data(writer, subscription_handles1[0]);
```

6.90.7 Tearing down a DataWriter

- See **Tearing Down An Entity** (p. 328) for eamples of the various ways to teardown a DataWriter

6.91 Subscriber Use Cases

The following #includes are needed for the examples on this page

```
#include <dds/sub/ddssub.hpp>
#include <dds/domain/ddsdomain.hpp>
#include <dds/core/ddscore.hpp>
```

6.91.1 Setting up a subscriber

- **Set up participant** (p. 307)
- Create a Subscriber with each of the available constructors

```
// Create the participant
dds::domain::DomainParticipant participant(0);
// Creating a Subscriber with default QoS and no listener
dds::sub::Subscriber subscriber1(participant);
// Creating a Subscriber with modified QoS
dds::sub::qos::SubscriberQos qos;
qos << dds::core::policy::Presentation::GroupAccessScope(true, false);
dds::sub::Subscriber subscriber2(participant, qos);
// Creating a Subscriber with modified QoS and a listener
auto listener = std::make_shared<ExampleSubscriberListener>();
dds::sub::Subscriber subscriber3(participant, qos, listener);
// Creating a Subscriber with modified QoS, a listener, and a sample_rejected()
// status mask
dds::sub::Subscriber subscriber4(
        participant,
        qos,
        listener,
        dds::core::status::StatusMask::sample_rejected());
```

6.91.2 Looking up Subscribers

- Lookup Subscribers

```
void howto_lookup_subscribers()
{
    dds::domain::DomainParticipant participant(0);
    // Create three Subscribers and retain them
    create_and_retain_subscriber(participant);
    create_and_retain_subscriber(participant);
    create_and_retain_subscriber(participant);
    // We don't have any references to the created Subscribers but we can look
    // them up using the participant that they were created with
    std::vector<dds::sub::Subscriber> subscribers1;
    // subscriber_count will equal 3 and subscribers1 will hold a reference to
    // each of the three subscribers that were created
    int subscriber_count =
        rti::sub::find_subscribers(participant, std::back_inserter(subscribers1));
    std::cout « "Subscriber count: " « subscriber_count « std::endl;
    // rti::sub::subscribers also takes a forward iterator and a max number of
    // subscribers to look up
    std::vector<dds::sub::Subscriber> subscribers2(2, dds::sub::Subscriber(dds::core::null));
    // subscriber_count will equal 2 because we're only asking for 2
    subscriber_count =
        rti::sub::find_subscribers(participant, subscribers2.begin(), 2);
    // We retained the subscribers so we must close them now
    for (std::vector<dds::sub::Subscriber>::iterator it = subscribers1.begin();
          it != subscribers1.end();
          it++) {
        (*it).close();
    }
}
void create_and_retain_subscriber(dds::domain::DomainParticipant &participant)
{
    dds::sub::Subscriber subscriber(participant);
    subscriber.retain();
}
```

6.91.3 Tearing down a subscriber

- See **Tearing Down An Entity** (p. 328) for eamples of the various ways to teardown a Subscriber

6.92 DataReader Use Cases

The following #includes are needed for the examples on this page

```
#include <dds/sub/ddssub.hpp>
#include <dds/domain/ddsdomain.hpp>
#include <dds/core/ddscore.hpp>
#include <dds/topic/ddstopic.hpp>
#include <dds/pub/ddspub.hpp>
#include "Foo.hpp"
```

6.92.1 Setting up a DataReader

- **Set up a Subscriber** (p. 315)
- **Set up a Topic** (p. 308)

- Create a DataReader of user-defined type **Foo** (p. 1588) with each of the available constructors

```
// Create the Participant
dds::domain::DomainParticipant participant(0);
// Create a Topic
dds::topic::Topic<Foo> topic(participant, "MyTopicName");
// When not explicitly provided to the constructor, the
// created DataReader's QoS is default, has a NULL listener, and a status mask
// dds::core::status::StatusMask::all()
// In all of the following constructors 'topic' could also be an instance
// of dds::topic::ContentFilteredTopic

// Creating a DataReader, creating the subscriber inline
dds::sub::DataReader<Foo> reader1(dds::sub::Subscriber(participant), topic);
// Create a Subscriber
dds::sub::Subscriber subscriber(participant);
// Create a DataReader with modified QoS
dds::sub::qos::DataReaderQos reader_qos =
        dds::core::QosProvider::Default().datareader_qos();
reader_qos << dds::core::policy::History::KeepAll();
dds::sub::DataReader<Foo> reader2(subscriber, topic, reader_qos);
// Create a DataReader with the Qos of a topic
reader_qos = topic.qos(); // Copy the policies in TopicQos into a DataReaderQos
dds::sub::DataReader<Foo> reader3(subscriber, topic, reader_qos);
// Create a DataReader with modified QoS and a listener
std::shared_ptr<ExampleDataReaderListener> listener =
        std::make_shared<ExampleDataReaderListener>();
dds::sub::DataReader<Foo> reader4(
        subscriber,
        topic,
        reader_qos,
        listener);
// Create a DataReader with modified QoS, a listener, and a
// liveliness_changed() status mask
dds::sub::DataReader<Foo> reader5(
        subscriber,
        topic,
        reader_qos,
        listener,
        dds::core::status::StatusMask::liveliness_changed());
```

6.92.2 Managing instances

- Getting an instance key value of user data type **Foo** (p. 1588)

```
// Instead of a sample of type Foo, key_value() could also take a
// TopicInstance<Foo> object. key_value only populates the key field in the
// sample
Foo retrieved_sample_key;
reader.key_value(retrieved_sample_key, handle);
```

6.92.3 Set up a DataReader to access received data

- **Set up a DataReader** (p. 316)
- Set up the DataReader to handle the DDS_DATA_AVAILABLE_STATUS status, in one or both of the following two ways.
- **Enable DDS_DATA_AVAILABLE_STATUS for the DDSDataReaderListener associated with the data reader** (p. 325)
  - The processing to handle the status change is done in the DataReaderListener on_data_available() method of the attached listener.
  - Typical processing will **access the received data** (p. 318).
- **Enable DDS_DATA_AVAILABLE_STATUS for the DDSStatusCondition associated with the data reader** (p. 327)

– The processing to handle the status change is done **when the data reader's attached status condition is triggered** (p. 330) and the DDS_DATA_AVAILABLE_STATUS status on the data reader is changed.
– Typical processing will **access the received data** (p. 318).

6.92.4 Accessing Received Data

6.92.4.1 Reading data samples

You can read data samples from a DataReader by receiving a loan, inspecting the samples and then returning the loan, or you can directly copy them.

- To iterate through loaned data samples in your application you will use a LoanedSamples container. The following code shows different ways to use this container

```
// Take all the samples in the reader queue (if you don't want to remove
// them from the queue and read them later again, use reader.read() instead)
dds::sub::LoanedSamples<Foo> samples = reader.take();
// LoanedSamples<Foo> is a container of LoanedSample<Foo> elements, which
// contain the actual data (Foo) and the SampleInfo
for (dds::sub::LoanedSamples<Foo>::iterator sample_it = samples.begin();
     sample_it != samples.end();
     ++sample_it) {
    if (sample_it->info().valid()) {
        std::cout « "Received: " « sample_it->data() « std::endl;
    }
}
// The previous code is simpler with a range-based for-loop (C++11)
for (auto sample : samples) {
    if (sample.info().valid()) {
        std::cout « "Received: " « sample.data() « std::endl;
    }
}
// If you don't need to access the SampleInfo in invalid-data samples,
// use valid_data() to simplify the for loop
auto valid_samples = rti::sub::valid_data(reader.take());
for (auto sample : valid_samples) {
    // No need to check sample.info().valid()
    std::cout « "Received: " « sample.data() « std::endl;
}
// You can also access samples using the subscript operator
if (samples.length() >= 2) {
    if (samples[1].info().valid()) {
        std::cout « "Second sample is " « samples[1].data() « std::endl;
    } else {
        std::cout « "Second sample is invalid\n";
    }
}
// You can copy the samples using std::copy:
std::vector<rti::sub::LoanedSample<Foo> > samples_copy;
std::copy(samples.begin(), samples.end(), std::back_inserter(samples_copy));
// You can also access the data directly. A sample can be implicitly
// converted into a reference to its data type as long as it contains
// valid data.
//
// The iterator returned by this other overload of valid_data provides
// another way to skip invalid-data samples
//
std::vector<Foo> data_copy;
std::copy(
    rti::sub::valid_data(samples.begin()),
    rti::sub::valid_data(samples.end()),
    std::back_inserter(data_copy));
// Attempting to access invalid data will result in an exception.
try {
    // The following will throw if !samples[0].info().valid()
    Foo data = samples[0].data();
    std::cout « data « std::endl;
    // The following will throw if any sample contains invalid data
    std::copy(samples.begin(), samples.end(), std::back_inserter(data_copy));
} catch (const dds::core::PreconditionNotMetError& ex) {
    std::cerr « "Attempted to access sample containing invalid data: "
```

```
            « ex.what() « std::endl;
}
// You can also apply other iterator-based algorithms,
// like this example that prints just one field of the data type
// (assuming that Foo contains an integer field called 'x')
std::transform(
    rti::sub::valid_data(samples.begin()),
    rti::sub::valid_data(samples.end()),
    std::ostream_iterator<int>(std::cout, ", "),
    [](const Foo& data) { return data.x(); });
// Once a LoanedSamples object goes out of scope its destructor
// automatically returns the loan to the middleware. If you need to
// explicitly return it, call return_loan()
//
// samples.return_loan()
```

- Although the LoanedSamples is the most generic way to access data in a DataReader, for convinience other overloaded versions of read/take allow to directly copy the data

```
// The first version of read/take receives a back-insert iterator
std::vector<dds::sub::Sample<Foo> > my_vector1;
reader.take(std::back_inserter(my_vector1));
// The second version receives a forward iterator and the maximum size
std::vector<dds::sub::Sample<Foo> > my_vector2(10);
reader.take(my_vector2.begin(), 10);
```

- See also **Working with IDL types** (p. 606)

6.92.4.2 Selecting what samples to read

The functions **read()** (p. 1024) and **take()** (p. 1024) select all samples in the DataReader queue. The function select() allows specifying different criteria about the data to read. A equivalent API, using the streaming >> operator is available too.

- select() example

```
// To select what samples you want to read (or take) from a DataReader
// queue, you can use a Selector. When you don't specify a parameter in the
// Selector, the default applies.
dds::sub::LoanedSamples<Foo> samples =
    // Call select() to create a Selector and then specify its parameters
    reader.select()
        // Specifies the maximum number of samples to read/take
        // in this function call. Default: set by Qos.
        .max_samples(20)
        // Specifies the sample, view and instance states - default: all.
        .state(dds::sub::status::DataState::new_instance())
        // Specifies a query on the data sample - default: all
        .content(dds::sub::Query(reader, "x > 10"))
        // Specifies what instance to read - default: all instances
        .instance(my_instance_handle)
        // And finally call read() or take()
        .take();
// You can also specify state and content by using a Condition. Reusing
// a Condition for multiple reads is more efficient than specifying the
// content() every time.
dds::sub::cond::QueryCondition query_condition(
    dds::sub::Query(reader, "x > 10"),
    dds::sub::status::DataState(
        dds::sub::status::SampleState::not_read(),
        dds::sub::status::ViewState::new_view(),
        dds::sub::status::InstanceState::alive()));
samples = reader.select()
    .condition(query_condition)
    .take();
// And you can read instance by instance using next_instance().
//
// To read the first instance, use InstanceHandle::nil()
dds::core::InstanceHandle previous_handle = dds::core::InstanceHandle::nil();
do {
    samples = reader.select()
        .next_instance(previous_handle)
        .take();
    if (samples.length() > 0) {
```

```
            // do something with samples ...
            // Update previous_handle to read the instance following this one
            previous_handle = samples[0].info().instance_handle();
        }
    } while (samples.length() > 0);
    // select() returns an object you can reuse
    dds::sub::DataReader<Foo>::Selector selector = reader.select().max_samples(20);
    samples = selector.read();
    // ...
    selector.state(dds::sub::status::DataState::new_instance());
    samples = selector.take();
```

- Streaming example

```
    using namespace dds::sub;
    // The overloaded shift operator defines a domain-specific language to
    // specify what samples to read. The following code has the same effect as
    // the first select() example from the previous code snippet
    LoanedSamples<Foo> samples;
    reader » take // first indicate if you want to read or take;
           » max_samples(20) // then specify the parameters you want...
           » state(status::DataState::new_instance())
           » content(Query(reader, "x > 10"))
           » instance(my_instance_handle)
           » samples; // and finally, the destination LoanedSamples container.
```

6.92.4.3 Reading samples using coherent access

If a Subscriber's PresentationQosPolicy access_scope is GROUP and ordered_access is true, the application can access the samples in order across DataWriters of the same group (Publisher with access_scope GROUP). The queue is ordered/sorted per topic across all instances belonging to DataWriters (or DataReaders) within the same Publisher (or Subscriber).

- Read samples using CoherentAccess

```
void howto_read_coherently()
{
    dds::domain::DomainParticipant participant(0);
    dds::topic::Topic<Foo> topic1(participant, "Topic1");
    dds::topic::Topic<Foo> topic2(participant, "Topic2");
    // Create a Subscriber with group presentation. The Publisher must also
    // have group presentation set in its QoS
    dds::sub::qos::SubscriberQos subscriber_qos;
    subscriber_qos « dds::core::policy::Presentation::GroupAccessScope(false, true);
    dds::sub::Subscriber subscriber(participant, subscriber_qos);
    dds::sub::DataReader<Foo> reader1(subscriber, topic1);
    dds::sub::DataReader<Foo> reader2(subscriber, topic2);
    // Assuming that a writer of "Topic1" wrote before the writer of "Topic2"
    // reader2 will only be able to access it's samples after reader1 takes
    // its sample as long as CoherentAccess to the samples has been started
    int sample_count = read_coherently(subscriber, reader2);
    std::cout « sample_count « std::endl; // will be 0
    // The CoherentAccess destructor called when read_coherently exited
    // will end coherent access to the samples.
    // You can also explicitly call coherent_access.end() to end coherent access
    // reader2 can now access its sample
    // ordered_samples.size() will = 1
    dds::sub::LoanedSamples<Foo> ordered_samples = reader2.take();
}
template <typename T>
int read_coherently(dds::sub::Subscriber& subscriber, dds::sub::DataReader<T>& reader)
{
    dds::sub::CoherentAccess coherent_access(subscriber);
    dds::sub::LoanedSamples<T> ordered_samples = reader.take();
    return ordered_samples.length();
}
```

6.92.5 DataReader Listeners

- Implementing a DataReaderListener class

```
// To create a DataReaderListener, inherit from the NoOpDataReaderListener class
// and then override any of the listener callback functions
class ExampleDataReaderListener : public dds::sub::NoOpDataReaderListener<Foo>
{
public:
    ExampleDataReaderListener() {}
public:
    void on_sample_lost(
       dds::sub::DataReader<Foo>&,
       const dds::core::status::SampleLostStatus &status)
    {
        std::cout « "on_sample_lost callback" « std::endl;
        // Access information from the status
        std::cout « "total_count = " « status.total_count() « std::endl;
        std::cout « "total_count_change = " « status.total_count_change() « std::endl;
    }
    void on_subscription_matched(
       dds::sub::DataReader<Foo>&,
       const dds::core::status::SubscriptionMatchedStatus &status)
    {
        std::cout « "on_subscription_matched callback" « std::endl;
        // Access information from the status
        std::cout « "total_count = " « status.total_count() « std::endl;
        std::cout « "total_count_change = " « status.total_count_change() « std::endl;
        std::cout « "current_count = " « status.current_count() « std::endl;
        std::cout « "current_count_change = " « status.current_count_change() « std::endl;
        dds::core::InstanceHandle handle = status.last_publication_handle();
        // Extension
        std::cout « "current_count_peak = "
                  « status.extensions().current_count_peak() « std::endl;
    }
    void on_sample_rejected(
       dds::sub::DataReader<Foo>&,
       const dds::core::status::SampleRejectedStatus &status)
    {
        std::cout « "on_sample_rejected callback" « std::endl;
        // Access information from the status
        std::cout « "total_count = " « status.total_count() « std::endl;
        std::cout « "total_count_change = " « status.total_count_change() « std::endl;
        std::cout « "last_reason = " « status.last_reason() « std::endl;
        dds::core::InstanceHandle handle = status.last_instance_handle();
    }
    void on_requested_incompatible_qos(
       dds::sub::DataReader<Foo>&,
       const dds::core::status::RequestedIncompatibleQosStatus &status)
    {
        std::cout « "on_requested_incompatible_qos callback" « std::endl;
        // Access information from the status
        std::cout « "total_count = " « status.total_count() « std::endl;
        std::cout « "total_count_change = " « status.total_count_change() « std::endl;
        std::cout « "last_policy_id = " « status.last_policy_id() « std::endl;
        dds::core::policy::QosPolicyCountSeq qos_seq = status.policies();
        if (qos_seq.size() > 0) {
            std::cout « "policy_id of one first incompatible Qos policy = "
                « qos_seq[0].policy_id() « std::endl;
        }
    }
    void on_requested_deadline_missed(
       dds::sub::DataReader<Foo>&,
       const dds::core::status::RequestedDeadlineMissedStatus &status)
    {
        std::cout « "on_requested_deadline_missed callback" « std::endl;
        // Access information from the status
        std::cout « "total_count = " « status.total_count() « std::endl;
        std::cout « "total_count_change = " « status.total_count_change() « std::endl;
        dds::core::InstanceHandle handle = status.last_instance_handle();
    }
    void on_liveliness_changed(
       dds::sub::DataReader<Foo>&,
       const dds::core::status::LivelinessChangedStatus &status)
    {
        std::cout « "on_liveliness_changed callback" « std::endl;
        // Access information from the status
        std::cout « "alive_count = " « status.alive_count() « std::endl;
        std::cout « "not_alive_count = " « status.not_alive_count() « std::endl;
        std::cout « "alive_count_change = " « status.alive_count_change() « std::endl;
        std::cout « "not_alive_count_change = " « status.not_alive_count_change() « std::endl;
        dds::core::InstanceHandle handle = status.last_publication_handle();
    }
    void on_data_available(dds::sub::DataReader<Foo> &reader)
    {
```

```
        std::cout « "on_data_available callback" « std::endl;
        // Read the available data
        dds::sub::LoanedSamples<Foo> samples = reader.read();
        // Print samples by copying to std::cout
        std::copy(
            samples.begin(),
            samples.end(),
            dds::sub::LoanedSamples<Foo>::ostream_iterator(std::cout, "\n"));
    }
};
```

- Setting a listener on a DataReader

```
dds::domain::DomainParticipant participant(0);
dds::topic::Topic<Foo> topic(participant, "Topic1");
auto listener = std::make_shared<ExampleDataReaderListener>();
// Create a DataReader with a listener and default
// dds::core::status::StatusMask::all()
dds::sub::DataReader<Foo> reader1(
   dds::sub::Subscriber(participant),
   topic,
   dds::sub::qos::DataReaderQos(),
   listener);
// Create a DataReader with a null listener
dds::sub::DataReader<Foo> reader2(
   dds::sub::Subscriber(participant),
   topic);
// Set the listener for the DataReader after creation
reader2.set_listener(
   listener,
   dds::core::status::StatusMask::sample_rejected());
// ...
// You can reset the listener to stop receiving status updates:
reader2.set_listener(nullptr);
```

6.92.6 Looking up DataReaders

- Lookup DataReaders by topic name

```
dds::domain::DomainParticipant participant(0);
dds::sub::Subscriber subscriber(participant);
dds::topic::Topic<Foo> topic1(participant, "Topic1");
dds::topic::Topic<Foo> topic2(participant, "Topic2");
// Create three DataReaders for topic1 and one for topic2 and retain them
create_and_retain_reader(subscriber, topic1);
create_and_retain_reader(subscriber, topic1);
create_and_retain_reader(subscriber, topic2);
// We don't have any references to the created DataReaders but we can look
// them up using the subscriber and topic name that they were created with
std::vector<dds::sub::DataReader<Foo> > readers;
// reader_count will equal 1. If more than one DataReader belongs to a
// subscriber with the same topic name then find may return any one of them
int reader_count =
    dds::sub::find<dds::sub::DataReader<Foo> >(
       subscriber,
       "Topic1",
       std::back_inserter(readers));
reader_count =
    dds::sub::find<dds::sub::DataReader<Foo> >(
       subscriber,
       "Topic2",
       std::back_inserter(readers));
```

- Lookup DataReaders by reader name

```
// Create a Subscriber with the name "MySubscriber"
dds::sub::qos::SubscriberQos subscriber_qos;
subscriber_qos « rti::core::policy::EntityName("MySubscriber");
dds::sub::Subscriber subscriber2(participant, subscriber_qos);
// Create a DataReader with the name "MyDataReader"
dds::sub::qos::DataReaderQos reader_qos;
reader_qos « rti::core::policy::EntityName("MyDataReader");
dds::sub::DataReader<Foo> reader(subscriber2, topic1, reader_qos);
// Retrieve the DataReader from the subscriber:
auto found_reader = rti::sub::find_datareader_by_name<dds::sub::DataReader<Foo> >(
```

```
        subscriber2,
        "MyDataReader");
assert(found_reader == reader);
// Retrieve the DataReader from the DomainParticipant, using also the
// name of the Subscriber:
found_reader = rti::sub::find_datareader_by_name<dds::sub::DataReader<Foo> >(
        participant,
        "MySubscriber::MyDataReader");
assert(found_reader == reader);
```

- Lookup all DataReaders

```
// To retrieve all the DataReaders at once, we use a collection of
// AnyDataReader. Each AnyDataReader may have a different template argument
std::vector<dds::sub::AnyDataReader> all_readers;
rti::sub::find_datareaders(subscriber, std::back_inserter(all_readers));
for (auto& r : all_readers) {
    // We can retrieve the typed DataReader if we know the type
    if (r.type_name() == "Foo") {
        dds::sub::DataReader<Foo> foo_reader = r.get<Foo>();
        // ...
    }
}
```

6.92.7 Getting Matched Publications

- Lookup a DataReader's matched publications

```
dds::domain::DomainParticipant participant(0);
dds::topic::Topic<Foo> topic(participant, "Topic1");
dds::sub::DataReader<Foo> reader(dds::sub::Subscriber(participant), topic);
// Create two writers that will match with the above reader
dds::pub::DataWriter<Foo> writer1(dds::pub::Publisher(participant), topic);
dds::pub::DataWriter<Foo> writer2(dds::pub::Publisher(participant), topic);
// Get a list of the InstanceHandles corresponding to the DataReader's
// matched publications.
dds::core::InstanceHandleSeq publication_handles1 =
    dds::sub::matched_publications(reader);
// matched_publications also takes in an iterator range and will return
// an InstanceHandle iterator to the end of the InstanceHandleSeq
dds::core::InstanceHandleSeq publication_handles2(2, dds::core::null);
dds::core::InstanceHandleSeq::iterator last =
    dds::sub::matched_publications(
       reader,
       publication_handles2.begin(),
       publication_handles2.end());
// now last == publication_handles2.end()
```

- Use the InstanceHandles obtained from **matched_publications()** (p. 1031) to get the PublicationBuiltinTopicData for the publications

```
dds::topic::PublicationBuiltinTopicData publication_data =
    dds::sub::matched_publication_data(reader, publication_handles1[0]);
```

6.92.8 Accessing the Built-in Subscriber and DataReaders

- The next two examples assume the following setup

```
// It is common to create entities disbaled when accessing and setting
// listeners on the built-in DataReaders so that information about
// discovery traffic is missed
dds::domain::qos::DomainParticipantFactoryQos qos;
qos << dds::core::policy::EntityFactory(false);
dds::domain::DomainParticipant::participant_factory_qos(qos);
dds::domain::DomainParticipant participant(0);
```

- Access the built-in Subscriber
```
dds::sub::Subscriber builtin_subscriber = dds::sub::builtin_subscriber(participant);
```

- Access the built-in DataReaders
```
// There are constants for each of the built-in topics accessible through
// the dds::topic namespace--see below for their use
// Access the built-in DataReader for ParticipantBuiltinTopicData
std::vector<dds::sub::DataReader<dds::topic::ParticipantBuiltinTopicData> > participant_reader;
int reader_count =
    dds::sub::find<dds::sub::DataReader<dds::topic::ParticipantBuiltinTopicData> >(
       builtin_subscriber,
       dds::topic::participant_topic_name(),
       std::back_inserter(participant_reader));
// Access the built-in DataReader for PublicationBuiltinTopicData
std::vector<dds::sub::DataReader<dds::topic::PublicationBuiltinTopicData> > publication_reader;
reader_count =
    dds::sub::find<dds::sub::DataReader<dds::topic::PublicationBuiltinTopicData> >(
       builtin_subscriber,
       dds::topic::publication_topic_name(),
       std::back_inserter(publication_reader));
// Access the built-in DataReader for SubscriptionBuiltinTopicData
std::vector<dds::sub::DataReader<dds::topic::SubscriptionBuiltinTopicData> > subscription_reader;
reader_count =
    dds::sub::find<dds::sub::DataReader<dds::topic::SubscriptionBuiltinTopicData> >(
       builtin_subscriber,
       dds::topic::subscription_topic_name(),
       std::back_inserter(subscription_reader));
// participant_reader[0], publication_reader[0] and subscription_reader[0]
// will now hold reference to the built-in DataReaders.
```

6.92.9 Using untyped DataReaders

When you want to hold a reference to a DataReader and the type T doesn't matter, you can use the class **dds::sub::AnyDataReader** (p. 807), which provides type-independent methods.

- For example, this function receives any DataReader
```
// This function can receive any DataReader, independently of the template types
void example_untyped(const dds::sub::AnyDataReader& untyped_reader)
{
    // Use type-independent operations
    std::cout << "The topic name is: " << untyped_reader.topic_name() << std::endl;
    if (untyped_reader.type_name() == "Foo") {
        // Get back the typed DataReader.
        //
        // If the reader's type is not Foo, this operation throws
        // dds::core::InvalidDowncastError.
        //
        // typed_reader and untyped_reader share references to the same object.
        dds::sub::DataReader<Foo> typed_reader = untyped_reader.get<Foo>();
    }
}
```

- And this is how you can call it
```
dds::sub::DataReader<Foo> reader(subscriber, topic);
example_untyped(reader); // automatic conversion to AnyDataReader
```

6.92.10 Tearing down a DataReader

- See **Tearing Down An Entity** (p. 328) for examples of the various ways to teardown a DataReader

6.93 Entity Use Cases

The following #includes are needed for the examples on this page

```
#include <dds/domain/ddsdomain.hpp>
```

6.93.1 Changing the QoS for an entity

The QoS for an entity can be specified at entity creation time. Once an entity has been created, its QoS can be manipulated as the following examples illustrate. A DataWriter is used in these examples, but the same can be applied to any <<***reference-type***>> *(*p. *351)*, such as DomainParticipants, Subscribers, Publishers, DataReaders, and Topics.

- See **QosProvider** (p. 604) for examples on how to manage user-defined QoS profiles.

- See **Qos Use Cases** (p. 602) for examples on how to work with the Qos policy classes.

- Get an entity's QoS settings

```
dds::pub::DataWriter<Foo> writer(dds::pub::Publisher(participant), a_topic);
// Get the DataWriter's QoS
dds::pub::qos::DataWriterQos writer_qos = writer.qos();
```

- Change the desired qos policy fields

```
// Change the History QoS for the DataWriter using the History QoS's
// KeepAll named constructor
writer_qos << dds::core::policy::History::KeepAll();
```

- Set the qos

```
// Set the modified QoS for the DataWriter
writer.qos(writer_qos);
```

6.93.2 Changing the listener and enabling/disabling statuses associated with it

The listener for an entity can be specified at the entity creation time. By default the listener is *enabled* for all the statuses supported by the entity.

Once an entity has been created, its listener and/or the statuses for which it is enabled can be manipulated as follows.

The following examples use a DomainParticipant but the same examples can be applied to all other entities (Subscribers, Publishers, DataReaders, DataWriters, and Topics)

- User defines entity listener methods

```
class ExampleDomainParticipantListener : public dds::domain::NoOpDomainParticipantListener
{
public:
    ExampleDomainParticipantListener() {}
public:
    void on_liveliness_lost(
       dds::pub::AnyDataWriter&,
       const::dds::core::status::LivelinessLostStatus&)
    {
        std::cout « "on_liveliness_lost" « std::endl;
    }
    void on_liveliness_changed(
       dds::sub::AnyDataReader&,
       const dds::core::status::LivelinessChangedStatus&)
    {
        std::cout « "on_liveliness_changed" « std::endl;
    }
    void on_inconsistent_topic(
       dds::topic::AnyTopic&,
       const dds::core::status::InconsistentTopicStatus&)
    {
        std::cout « "on_inconsistent_topic" « std::endl;
    }
    void on_data_on_readers(
       dds::sub::Subscriber&)
    {
        std::cout « "on_data_on_readers" « std::endl;
    }
    void on_offered_deadline_missed(
       dds::pub::AnyDataWriter&,
       const::dds::core::status::OfferedDeadlineMissedStatus&)
    {
        std::cout « "on_offered_deadline_missed" « std::endl;
    }
    void on_offered_incompatible_qos(
       dds::pub::AnyDataWriter&,
       const::dds::core::status::OfferedIncompatibleQosStatus&)
    {
        std::cout « "on_offered_incompatible_qos" « std::endl;
    }
    void on_publication_matched(
       dds::pub::AnyDataWriter&,
       const::dds::core::status::PublicationMatchedStatus&)
    {
        std::cout « "on_publication_matched" « std::endl;
    }
    void on_requested_deadline_missed(
       dds::sub::AnyDataReader&,
       const dds::core::status::RequestedDeadlineMissedStatus&)
    {
        std::cout « "on_requested_deadline_missed" « std::endl;
    }
    void on_requested_incompatible_qos(
       dds::sub::AnyDataReader&,
       const dds::core::status::RequestedIncompatibleQosStatus&)
    {
        std::cout « "on_requested_incompatible_qos" « std::endl;
    }
    void on_sample_lost(
       dds::sub::AnyDataReader&,
       const dds::core::status::SampleLostStatus&)
    {
        std::cout « "on_sample_lost" « std::endl;
    }
    void on_sample_rejected(
       dds::sub::AnyDataReader&,
       const dds::core::status::SampleRejectedStatus&)
    {
        std::cout « "on_sample_rejected" « std::endl;
    }
    void on_subscription_matched(
       dds::sub::AnyDataReader&,
       const dds::core::status::SubscriptionMatchedStatus&)
    {
        std::cout « "on_subscription_matched" « std::endl;
    }
    void on_data_available(
       dds::sub::AnyDataReader&)
    {
        std::cout « "on_data_available" « std::endl;
```

```
    }
};
```

- Get an entity's listener
```
ExampleDomainParticipantListener *listener = new ExampleDomainParticipantListener;
dds::domain::DomainParticipant participant(
   0, dds::domain::qos::DomainParticipantQos(), listener);
dds::domain::DomainParticipantListener *retrieved_listener = participant.listener();
```

- Enable statuses for the `listener`
```
// The default constructor creates an empty StatusMask
dds::core::status::StatusMask enabled_status_list;
// ...
// Add more statuses to the enabled_status_list
enabled_status_list |= dds::core::status::StatusMask::liveliness_changed() |
    dds::core::status::StatusMask::offered_incompatible_qos();
```

- Disable a status for the `listener`
```
enabled_status_list &= ~dds::core::status::StatusMask::offered_incompatible_qos();
```

- Set an entity's listener and only enable the listener for the statuses specified by the `enabled_status_list`.
```
participant.listener(listener, enabled_status_list);
```

6.93.3 Enabling/Disabling statuses associated with a status condition

Upon entity creation, by default, all the statuses are *enabled* for the DDS_StatusCondition associated with the entity.

Once an entity has been created, the list of statuses for which the DDS_StatusCondition is triggered can be manipulated as follows.

- Given an `entity` and the associated `status_condition`
```
ExampleDomainParticipantListener *listener = new ExampleDomainParticipantListener;
dds::domain::DomainParticipant participant(
   0, dds::domain::qos::DomainParticipantQos(), listener);
dds::core::cond::StatusCondition status_condition(participant);
```

- Get the list of statuses enabled for the `status_condition`
```
dds::core::status::StatusMask enabled_status_list = status_condition.enabled_statuses();
```

- Check if a given status is enabled for the `status_condition`
```
// StatusMask inherits from std::bitset. You can use std::bitset's any()
// method to check if a given status is enabled
if ((enabled_status_list & dds::core::status::StatusMask::data_on_readers()).any()) {
    // Do something to handle the on_data_on_readers status...
}
```

- Enable statuses for the `status_condition`

```
status_condition.enabled_statuses(
   dds::core::status::StatusMask::inconsistent_topic() |
   dds::core::status::StatusMask::sample_rejected());
```

- Disable statuses for the `status_condition`

```
status_condition.enabled_statuses(
   ~dds::core::status::StatusMask::inconsistent_topic());
```

6.93.4 Ignoring Entities

A Topic is used in this example, but each of the other entities (DomainParticipants, Publishers, Subscribers, and Data↩ Writers) also have an **ignore()** (p. 1359) method which takes the DomainParticipant and InstanceHandle(s) of the entities that are to be ignored

- Ignore Topics

```
dds::domain::DomainParticipant participant(0);
dds::topic::Topic<Foo> topic1(participant, "Topic1");
dds::topic::Topic<Foo> topic2(participant, "Topic2");
dds::topic::Topic<Foo> topic3(participant, "Topic3");
// This participant will now ignore this topic
dds::topic::ignore(participant, topic1.instance_handle());
dds::core::InstanceHandleSeq handles;
handles.push_back(topic2.instance_handle());
handles.push_back(topic3.instance_handle());
// This participant will now ignore the other two topics too
dds::topic::ignore(participant, handles.begin(), handles.end());
```

6.93.5 Tearing Down An Entity

The following examples use a DomainParticipant to show the various patterns which exist to tear down an entity, but the same examples can also be applied when shutting down Subscribers, Publishers, DataReaders, DataWriters, and Topics

- Let all references to a given entity go out-of-scope and let the destructor take care of entity clean up

```
{
    {
        // Create a participant on domain 0
        dds::domain::DomainParticipant participant(0);

        // ...
    } // participant destroyed here
    // found_participant will equal dds::core::null because the participant's
    // destructor was called when the only reference to it went out of scope
    dds::domain::DomainParticipant found_participant = dds::domain::find(0);
}
```

- Explicitly call **close()** (p. 1024) on one of the existing references to an Entity

```
// Create a participant on domain 0
dds::domain::DomainParticipant participant(0);
// All Entities are Reference types, meaning that the assignment operator
// will not make a copy of the participant, but instead will just create
// another reference to the previously created participant
dds::domain::DomainParticipant same_participant = participant;
// Call close() on same_participant to explicitly destroy the participant.
// Both references (participant and same_participant) are now invalid.
same_participant.close();
```

```
    try {
        // Any attempt to use the closed participant, will cause a
        // dds::core::AlreadyClosedError exception
        participant.enable();
    }
    catch (dds::core::AlreadyClosedError& ex) {
        std::cout « "Expected AlreadyClosedError: " « ex.what() « std::endl;
    }
```

- Assigning **dds::core::null** (p. 438) to a variable has the same effect as if the reference went ouf of scope. Assigning **dds::core::null** (p. 438) to the last reference destroys the underlying object.

```
    // Create a reference to a null participant
    dds::domain::DomainParticipant participant1(dds::core::null);
    // Now create a participant on domain 0
    dds::domain::DomainParticipant participant2(0);
    // Assign participant1 to this valid participant
    participant1 = participant2;
    // Assign dds::core:null to participant2. The participant will not be
    // destroyed because participant1 is still a valid reference
    participant2 = dds::core::null;
    // We can still use the participant because participant1 is still a
    // valid reference
    dds::domain::qos::DomainParticipantQos qos =
        participant1.default_participant_qos();
    // Assigning dds::core::null to the only remaining reference to the
    // participant will now cause its destructor to be called
    participant1 = dds::core::null;
    // found_participant will equal dds::core::null because the previous
    // assignment to dds::core::null caused the participant to be destroyed
    dds::domain::DomainParticipant found_participant = dds::domain::find(0);
```

- Retaining and then destroying an Entity

```
void howto_teardown_retain()
{
    // To prevent participant destruction when all references go out of scope
    // you can call retain().
    const uint32_t domain_id_0 = 0;
    const uint32_t domain_id_1 = 1;
    create_and_retain_participant(domain_id_0, false);
    create_and_retain_participant(domain_id_1, true);
    // We did not retain the participant with domain_id_0, so the following
    // will result in participant0 == dds::core::null
    dds::domain::DomainParticipant participant0 = dds::domain::find(domain_id_0);
    // We can recover the participant with domain_id_1 using find() because we
    // retained it. participant1 will be a valid reference to
    // the participant that was created on domain 1
    dds::domain::DomainParticipant participant1 = dds::domain::find(domain_id_1);
    // Because we called retain on the participant, the only way to destroy
    // the participant is to explicitly call close() on a reference to the
    // participant. Any attempt to use the participant after calling
    // close will cause a dds::core::AlreadyClosedError exception to be thrown
    // Destroy the participant
    participant1.close();
    try {
        // This call will cause a dds::core::AlreadyClosedError exception
        participant1.enable();
    }
    catch (dds::core::AlreadyClosedError& ex) {
        std::cout « "Expected AlreadyClosedError: " « ex.what() « std::endl;
    }

}
void create_and_retain_participant(const uint32_t domain_id, bool retain)
{
    // Create and retain a participant. If retained, the participant will not
    // be destroyed when it goes out of scope and can be retrieved again
    dds::domain::DomainParticipant participant(domain_id);
    if (retain) {
        participant.retain();
    }
}
```

6.94 Waitset Use Cases

Using WaitSets and Conditions.

Using WaitSets and Conditions.

The following #includes are needed for the examples on this page

```
#include <iostream>
#include <dds/core/ddscore.hpp>
#include <dds/sub/ddssub.hpp>
```

6.94.1 Setting up a WaitSet

- Create a WaitSet and attach Conditions

```
// Create a WaitSet
dds::core::cond::WaitSet waitset;
// Create a GuardCondition
dds::core::cond::GuardCondition guard_cond;
// Create a StatusCondition for a given Entity
dds::core::cond::StatusCondition status_cond(entity);
// Create a ReadCondition for a reader with a specific DataState
dds::sub::cond::ReadCondition read_cond(
    reader, dds::sub::status::DataState(
        dds::sub::status::SampleState::not_read(),
        dds::sub::status::ViewState::any(),
        dds::sub::status::InstanceState::any()));
// Attach conditions
waitset += guard_cond; // using += operator
waitset += status_cond;
waitset.attach_condition(read_cond); // or using attach_condition()
```

6.94.2 Waiting for Condition(s) to trigger

You have two ways to wait for Conditions to trigger: wait or dispatch.

- Wait example

```
// Wait for at most 20 seconds until one or more conditions are active and
// then return them.
// In case of timeout, active_conditions will be empty.
dds::core::cond::WaitSet::ConditionSeq active_conditions =
    waitset.wait(dds::core::Duration::from_secs(20));
for (uint32_t i = 0; i < active_conditions.size(); i++) {
    if (active_conditions[i] == guard_cond) {
        std::cout « "guard_cond was triggered\n";
    } else if (active_conditions[i] == status_cond) {
        std::cout « "status_cond was triggered\n";
    } // ...
}
```

- Dispatch example

```
using dds::core::cond::Condition;
using dds::sub::cond::ReadCondition;
// create a WaitSet
dds::core::cond::WaitSet waitset;
// Create a ReadCondition for reader1 with a functor handler
ReadCondition read_cond1(reader1, dds::sub::status::DataState::any(), []() {
    std::cout « "read_cond1 was triggered\n";
});
ReadCondition read_cond2(
        reader2,
        dds::sub::status::DataState::any(),
        [](Condition c) {  // The handler can optionally
                           // receive the condition
            auto rc = dds::core::polymorphic_cast<ReadCondition>(c);
            auto reader = rc.data_reader();  // reader == reader2
            std::cout « "read_cond2 was triggered (topic is "
                      « reader.topic_name() « ")" « std::endl;
        });
// Attach conditions
waitset += read_cond1;
waitset += read_cond2;
// Wait at most 20 seconds until one or more conditions are active and
// then call the handler of the active conditions. You can repeat this in
// a loop.
waitset.dispatch(dds::core::Duration::from_secs(20));
```

See also

Filtering with Query Conditions (p. 332)

6.95 Filter Use Cases

Working with data filters.

Working with data filters.

6.95.1 Introduction

RTI Connext supports filtering data either during the exchange from **dds::pub::DataWriter** (p. 1138) to **dds::sub::DataReader** (p. 976), or after the data has been stored at the **dds::sub::DataReader** (p. 976).

Filtering during the exchange process is performed by a **dds::topic::ContentFilteredTopic** (p. 954), which is created by the **dds::sub::DataReader** (p. 976) as a way of specifying a subset of the data samples that it wishes to receive.

Filtering samples that have already been received by the **dds::sub::DataReader** (p. 976) is performed by creating a **dds::sub::cond::QueryCondition** (p. 2059), which can then used to check for matching samples, be alerted when matching samples arrive, or retrieve matching samples.

Filtering may be performed on any topic, either keyed or un-keyed, except the built-in topics. Filtering may be performed on any field, subset of fields, or combination of fields, subject only to the limitations of the filter syntax.

6.95.2 Code Examples

The following #includes are needed for the examples on this page

```
#include <iostream>
#include <dds/topic/ddstopic.hpp>
#include <dds/sub/ddssub.hpp>
#include "Foo.hpp"
```

6.95.3 Filtering with ContentFilteredTopic

- **Set up a Subscriber** (p. 315)
- **Set up a Topic** (p. 308)
- Create a ContentFilteredTopic, of user data type **`Foo`** (p. 1588):

```
// Create the DomainParticipant
dds::domain::DomainParticipant participant(0);
// Create a Topic
dds::topic::Topic<Foo> topic(participant, "MyTopicName");
// Create the parameter list
std::vector<std::string> cft_parameters(2);
cft_parameters[0] = "1";
cft_parameters[1] = "100";
// Create the ContentFilteredTopic
dds::topic::ContentFilteredTopic<Foo> cft(
    topic, // related topic
    "ContentFilteredTopic", // local name for the CFT
    dds::topic::Filter( // filter (constructed in-line in this example)
        "x > %0 AND x < %1", // expression
        cft_parameters)); // parameter vector
// Parameters are optional
dds::topic::ContentFilteredTopic<Foo> cft2(
    topic,
    "ContentFilteredTopic2",
    dds::topic::Filter("x = 100 AND y < 20"));
```

- Create a DataReader using the ContentFilteredTopic:

```
dds::sub::DataReader<Foo> reader(subscriber, cft);
```

  Once setup, reading samples with a **`dds::topic::ContentFilteredTopic`** (p. 954) is exactly the same as normal reads or takes, as described in **DataReader Use Cases** (p. 316).

- Changing filter criteria:

```
std::vector<std::string> new_cft_parameters(2);
new_cft_parameters[0] = "5";
new_cft_parameters[1] = "9";
cft.filter_parameters(new_cft_parameters.begin(), new_cft_parameters.end());
```

6.95.4 Filtering with Query Conditions

- Creating a QueryCondition

```
// Create parameter list
std::vector<std::string> query_parameters(2);
query_parameters[0] = "1";
query_parameters[1] = "100";
// Create query condition
dds::sub::cond::QueryCondition query_condition(
    dds::sub::Query(reader, "x > %0 AND x < %1", query_parameters),
    dds::sub::status::DataState::any_data());
```

- You can use a QueryCondition in a WaitSet: see **Waiting for Condition(s) to trigger** (p. 330)
- And to query for data in a DataReader: see **Selecting what samples to read** (p. 319)
- To modify the filter criteria you can use **dds::sub::cond::QueryCondition::parameters** (p. 2062) (similar to **dds::topic::ContentFilteredTopic::filter_parameters** (p. 956))
- This example shows how to create a condition with a handler that takes the samples received by a DataReader and selected by the QueryCondition's filter.

```
using dds::sub::cond::QueryCondition;

QueryCondition query_condition2(
    dds::sub::Query(reader, "x > %0 AND x < %1", query_parameters),
    dds::sub::status::DataState::any_data(),
    // The handler function singature allows receiving the related condition
    // as a parameter (it also allows zero arguments).
    [&reader](dds::core::cond::Condition condition)
    {
        // Downcast a Condition (a reference type) into a QueryCondition
        auto condition_as_qc =
                dds::core::polymorphic_cast<QueryCondition>(condition);
        // Use the condition's filter to select specific samples
        auto samples = reader.select().condition(condition_as_qc).take();
        for (auto& sample : samples) {
            std::cout << "Sample received: " << sample << std::endl;
        }
    });
```

6.95.5 Filtering Performance

Although RTI Connext supports filtering on any field or combination of fields using the SQL syntax of the built-in filter, filters for keyed topics that filter solely on the contents of key fields have the potential for much higher performance. This is because for key-only filters, the **dds::sub::DataReader** (p. 976) caches the results of the filter (pass or not pass) for each instance. When another sample of the same instance is seen at the **dds::sub::DataReader** (p. 976), the filter results are retrieved from the cache, dispensing with the need to call the filter function.

This optimization applies to all filtering using the built-in SQL filter, performed by the **dds::sub::DataReader** (p. 976), for either **dds::topic::ContentFilteredTopic** (p. 954) or **dds::sub::cond::QueryCondition** (p. 2059). This does *not* apply to filtering perfomed for **dds::topic::ContentFilteredTopic** (p. 954) by the **dds::pub::DataWriter** (p. 1138).

6.96 Creating Custom Content Filters

Creating a custom content filter.

Creating a custom content filter.

6.96.1 Introduction

By default, RTI Connext creates content filters with the DDS SQL filter, which implements a superset of the DDS-specified SQL WHERE clause. However, in many cases this filter may not be what you want. Some examples are:

- The default filter can only filter based on the content of a sample, not on a computation on the content of a sample. You can use a custom filter that is customized for a specific type and can filter based on a computation of the type members.
- You want to use a different filter language then SQL

This HOW-TO explains how to write your own custom filter and is divided into the following sections:

- **The Custom Content Filter API** (p. 334)
- **Example Custom Writer Content Filter** (p. 336)

6.96.2 The Custom Content Filter API

A custom content filter is created by calling the **dds::domain::DomainParticipant::register_contentfilter** (p. 1349) function with a **rti::topic::CustomFilter** (p. 967). A CustomFilter is created with either a **rti::topic::ContentFilter** (p. 950) or **rti::topic::WriterContentFilter** (p. 2652).

A ContentFilter contains a **compile**, an **evaluate** and a **finalize** function.

A WriterContentFilter contains a **compile**, an **evaluate**, a **finalize**, a **writer_attach**, **writer_compile**, **writer_evaluate**, **wrtier_detach**, and **writer_finalize** functions.

To use a custom filter in a **dds::topic::ContentFilteredTopic** (p. 954) or a dds::sub::QueryCondition, the name used to register it with the participant has to be set in the **dds::topic::Filter** (p. 1558)
or in the **dds::sub::Query** (p. 2053) that is used to create the corresponding ContentFilteredTopic or QueryCondition.

A custom ContentFilter is used by RTI Connext at the following times during the life-time of a ContentFilteredTopic (the function called is shown in parenthesis).

- When a ContentFilteredTopic is created (**compile** (p. 334))
- When the filter parameters are changed on the ContentFilteredTopic (**compile** (p. 334))
- When a sample is filtered (**evaluate** (p. 335)). This function is called by the RTI Connext core with a de-serialized sample
- When a ContentFilteredTopic is deleted (**finalize** (p. 335))

A custom WriterContentFilter is used by RTI Connext at the following times during the life-time of a ContentFilteredTopic (the function called is shown in parenthesis).

- When a DataWriter discovers a DataReader with a ContentFilteredTopic or when a DataWriter is notified of a change in the DataReader's filter parameter (**writer_compile** (p. 335))
- When the DataWriter matches a DataReader using the specified filter for the first time (**writer_attach** (p. 335))
- When a DataWriter writes a new sample (**writer_evaluate** (p. 335)). This function is called by the RTI Connext core with a de-serialized sample
- When RTI Connext is finished using the sequence of Cookies returned by writer_evaluate (**writer_return_loan** (p. 335))
- When a DataWriter is no longer matching with a DataReader for which it was previously performing writer-side filtering (**writer_finalize** (p. 336))

6.96.2.1 The compile function

The "compile" function is used to **compile** a filter expression and expression parameters. Please note that the term **compile** is intentionally loosely defined. It is up to the user to decide what this function should do and return.

6.96.2.2 The evaluate function

When using a ContentFilter, the "evaluate" function is called each time a sample is received to determine if a sample should be filtered out and discarded.

When using a WriterContentFilter, the "evaluate" function is called each time a sample is written to determine if a sample should be filtered out and discarded. It is called for each DataReader for which the DataWriter is filtering and for which the writer_compile function set the ExpressionProperty.writer_side_filter_optimization to false.

6.96.2.3 The finalize function

The "finalize" function is called when an instance of the custom content filter is no longer needed. When this function is called, it is safe to free all resources used by this particular instance of the custom content filter.

6.96.2.4 The writer_attach function

The "writer_attach" function is called the first time that a DataWriter matches with a DataReader with the same Content↩ Filter. It will not be called for subsequent DataReaders that are using the same filter. This function is used to create some state required to perform filtering on the writer-side. It is entirely up to you, as the implementer of the filter, to decide if the filter requires this state.

6.96.2.5 The writer_compile function

The "writer_compile" function is called when a DataWriter matches with a DataReader with the same ContentFilter. It is called every time that the DataWriter matches a DataReader that is using the same filter as well as each time the DataWriter is notified that a DataReader's filter parameters have changed. This function will receive as an input a **rti::core::Cookie** (p. 965) which uniquely identifies the DataReader for which the function was invoked.

6.96.2.6 The writer_evaluate function

The "writer_evaluate" function is called every time that a DataWriter writes a new sample. Its purpose is to evaluate the sample for all the readers for which the DataWriter is performing writer-side filtering and return the sequence of **rti::core::Cookie** (p. 965) associated with the DataReaders whose filter pass the sample.

6.96.2.7 The writer_detach function

The "writer_detach" function is called when an instance of the custom content filter is no longer needed. When this function is called, it is safe to free all resources used by this particular instance of the custom content filter.

6.96.2.8 The writer_return_loan function

The "writer_return_loan" function is called to return the loan on the rti::core::CookieSeq provided by the writer_evaluate function.

6.96.2.9 The writer_finalize function

The "writer_finalize" function will be called by Connext to notify the filter implementation that the DataWriter is no longer matching with a DataReader for which it was previously performing writer-side filtering. This will allow the filter to purge any state it was maintaining for the DataReader.

6.96.3 Example Custom Writer Content Filter

Assume that you have a type **Foo** (p. 1588).

Our filter will show how to enable the writer-side filter optimization for some readers and not for others. The ones with writer-side filter optimization will only pass samples where Foo.x == y where y is a value determined by an expression parameter, see the writer_evaluate function. Readers without the optimization turned on will pass all samples where Foo.x $>$ 7, see the evaluate function. The filter will **only** be used to filter samples of type **Foo** (p. 1588).

The following #includes are needed for the examples on this page

```
#include <iostream>
#include <dds/topic/ddstopic.hpp>
#include <rti/topic/findImpl.hpp>
#include <dds/sub/ddssub.hpp>
#include <dds/pub/ddspub.hpp>
#include <dds/core/ddscore.hpp>
#include "Foo.hpp"
```

The following is the definition of the WriterFilterData, the state that is created and returned by the writer_attach method:

```
class WriterFilterData
{
public:
    typedef std::pair<rti::core::Cookie, int32_t> CookieValue;
    typedef std::vector<CookieValue> CookieValueSeq;
public:
    WriterFilterData()
    {
    }
    void add_pair(const CookieValue& cookie_value_pair)
    {
        reader_pairs_.push_back(cookie_value_pair);
    }
    CookieValueSeq& reader_pairs()
    {
        return reader_pairs_;
    }
private:
    CookieValueSeq reader_pairs_;
};
```

And here is the declaration of our custom writer content filter. Notice, we are inheriting from **rti::topic::WriterContentFilter** (p. 2652). We could have inherited from **rti::topic::ContentFilter** (p. 950) or **rti::topic::WriterContentFilterHelper** (p. 2657) here too to create other custom content filters.

```
// A custom WriterContentFilter
class ExampleWriterContentFilter :
        public rti::topic::WriterContentFilter<
            Foo,
            rti::topic::no_compile_data_t,
            WriterFilterData>
{
public:
    ExampleWriterContentFilter() {}
    ~ExampleWriterContentFilter() {}
    rti::topic::no_compile_data_t& compile(
       const std::string& expression,
       const dds::core::StringSeq& parameters,
       const dds::core::optional<dds::core::xtypes::DynamicType>& type_code,
       const std::string& type_class_name,
       rti::topic::no_compile_data_t* old_compile_data);
    bool evaluate(
       rti::topic::no_compile_data_t& compile_data,
```

```
       const Foo& sample,
       const rti::topic::FilterSampleInfo& meta_data);
    void finalize(rti::topic::no_compile_data_t& compile_data);
    WriterFilterData& writer_attach();
    void writer_compile(
        WriterFilterData& writer_filter_data,
        rti::topic::ExpressionProperty& prop,
        const std::string& expression,
        const dds::core::StringSeq& parameters,
        const dds::core::optional<dds::core::xtypes::DynamicType>& type_code,
        const std::string& type_class_name,
        const rti::core::Cookie& cookie);
    rti::core::CookieSeq& writer_evaluate(
       WriterFilterData& writer_filter_data,
       const Foo& sample,
       const rti::topic::FilterSampleInfo &meta_data);
    void writer_finalize(
        WriterFilterData& writer_filter_data,
        const rti::core::Cookie& cookie);
    void writer_detach(WriterFilterData& writer_filter_data);
    void writer_return_loan(
        WriterFilterData& writer_filter_data,
        rti::core::CookieSeq& cookies);
    void writer_data(const WriterFilterData& is_writer_data) { writer_data_ = is_writer_data; }
    WriterFilterData& writer_data() { return writer_data_; }
    void reset_cookie_seq()
    {
        cookie_seq_.clear();
    }
    void add_cookie(rti::core::Cookie& cookie)
    {
        cookie_seq_.resize(cookie_seq_.size() + 1);
        cookie_seq_[cookie_seq_.size() - 1] = cookie;
    }
    rti::core::CookieSeq& cookie_seq()
    {
        return cookie_seq_;
    }
private:
    rti::core::CookieSeq cookie_seq_;
    WriterFilterData writer_data_;
};
```

6.96.3.1 Writing the Compile Function

Since we already know what the expression is (Foo.x $>$ 7), we can simply return rti::topic::no_compile_data.

Below is the entire **compile** (p. 334) function.

```
rti::topic::no_compile_data_t& ExampleWriterContentFilter::compile(
       const std::string&,
       const dds::core::StringSeq&,
       const dds::core::optional<dds::core::xtypes::DynamicType>&,
       const std::string&,
       rti::topic::no_compile_data_t*)
{
    // We don't have any compile data to setup in this compile function. We
    // will do all necessary setup in the writer_compile function
    return rti::topic::no_compile_data;
}
```

6.96.3.2 Writing the Evaluate Function

The next step is to implement the **evaluate** function. The evaluate function receives no_compile_data because it is unecessary in this example. The function then evaluates the received sample against our filter expression and passes the sample if Foo.x $>$ 7. Below is the entire **evaluate** (p. 335) function.

```
bool ExampleWriterContentFilter::evaluate(
   rti::topic::no_compile_data_t&,
   const Foo& sample,
   const rti::topic::FilterSampleInfo&)
{
```

```
    if (sample.x() > 7) {
        return true;
    }
    return false;
}
```

6.96.3.3 Writing the Finalize Function

The next function to write is the finalize function. It is safe to free all resources used by this particular instance of the custom content filter that is allocated in **compile**. Because we did not create any resources in the **compile** function, we have nothing to do in the **finalize**. Below is the entire **finalize** (p. 335) function.

```
void ExampleWriterContentFilter::finalize(
   rti::topic::no_compile_data_t&)
{
    // There is nothing to finalize in this example
}
```

6.96.3.4 Writing the Writer Attach Function

The **writer_attach** is used to create some state required to perform filtering on the writer-side. In our example filter, this state is kept as part of our custom filter class, ExampleWriterContentFilter, so we therefore simply return a reference to our writer data that will be used during the **writer_compile** and **writer_evaluate** functions.

```
WriterFilterData& ExampleWriterContentFilter::writer_attach()
{
    // Setup the writer_filter_data to point to our WriterFilterData
    return writer_data();
}
```

6.96.3.5 Writing the Writer Compile Function

The **writer_compile** function is called when a DataWriter matches with a DataReader with the same ContentFilter. In our case, we use the parameters to determine if we should turn on the writer-side filtering optimization. If parameters[0] == 1, then we store the provided **rti::core::Cookie** (p. 965) along with parameters[1] in our writer_filter_data to be accessed during the writer_evaluate function whenever we receive a new sample.

```
void ExampleWriterContentFilter::writer_compile(
   WriterFilterData& writer_filter_data,
   rti::topic::ExpressionProperty& prop,
   const std::string& /* expression */,
   const dds::core::StringSeq& parameters,
   const dds::core::optional<dds::core::xtypes::DynamicType>& /* type */,
   const std::string& /* type_class_name */,
   const rti::core::Cookie& cookie)
{
    int length = parameters.size();
    bool optimize = false;
    int32_t x = 0;
    if (length == 2) {
        std::istringstream(parameters[0]) >> optimize;
        std::istringstream(parameters[1]) >> x;
    }
    // Add this DataReader's cookie to our writer filter data
    if (optimize) {
        writer_filter_data.add_pair(std::make_pair(cookie, x));
    }
    prop.writer_side_filter_optimization(optimize);
}
```

6.96.3.6 Writing the Writer Evaluate Function

The writer_evaluate function receives our stored writer_filter_data and a sample to evaluate. We iterate through our (**rti::core::Cookie** (p. 965), value) pairs and add any Cookie with a matching value of x to the sample's Foo.x to the rti::core::CookieSeq that we return. Any Cookie in this sequence then represents a DataReader to which this sample will be sent.

```
rti::core::CookieSeq& ExampleWriterContentFilter::writer_evaluate(
   WriterFilterData& writer_filter_data,
   const Foo& sample,
   const rti::topic::FilterSampleInfo&)
{
    WriterFilterData::CookieValueSeq& reader_pairs =
           writer_filter_data.reader_pairs();
    // If x == value, pass the reader
    for (uint32_t i = 0; i < reader_pairs.size(); i++) {
        if (sample.x() == reader_pairs[i].second) {
            add_cookie(reader_pairs[i].first);
        }
    }
    return cookie_seq();
}
```

6.96.3.7 Writing the Writer Detach Function

It is safe to free all resources used by this particular instance of the custom content filter that is allocated in **writer_attach**. Because we did not allocate any resources in the **writer_attach** function, there is nothing to release in the writer_finalize. Below is the entire **writer_detach** (p. 335) function.

```
void ExampleWriterContentFilter::writer_detach(
   WriterFilterData&)
{
    // Nothing to do in writer detach for this example
}
```

6.96.3.8 Writing the Writer Return Loan Function

RTI Connext uses the writer_return_loan function specified in the WriterContentFilter to indicate to the filter implementation that it has finished using the sequence of cookies returned by the filter writer_evaluate function. Your filter implementation should not free the memory associated with the cookie sequence before the writer_return_loan function is called. You can also create your custom content filter by inheriting from the **rti::topic::WriterContentFilterHelper** (p. 2657), which manages the DataReader CookieSeq for you. If you do that, then the writer_return_loan is implemented for you. Below is the entire **writer_return_loan** (p. 335) function.

```
void ExampleWriterContentFilter::writer_return_loan(
    WriterFilterData&,
    rti::core::CookieSeq&)
{
    // The cookie sequence's destructor will take care of returning the loan
    // to the middleware, but we will reset the size to 0
    const_cast<rti::core::CookieSeq&>(cookie_seq()).resize(0);
}
```

6.96.3.9 Writing the Writer Finalize Function

The writer_finalize function specified in the WriterContentFilter will be called when the DataWriter no longer matches with a DataReader that was created with ContentFilteredTopic. This will allow the filter implementation to delete any state it was maintaining for the DataReader. Because we did not create any resources in the **writer_compile** function, we have nothing to do in the **writer_finalize**. Below is the entire **writer_finalize** (p. 336) function.

```
void ExampleWriterContentFilter::writer_finalize(
   WriterFilterData&,
   const rti::core::Cookie&)
{
    // There is nothing to finalize in this example
}
```

6.96.3.10 Creating a CustomFilter

The first thing that an application needs to do when using a custom filter is to give it a name and wrap an instance of their custom filter with a **rti::topic::CustomFilter** (p. 967). The CustomFilter class receives shared pointer to an instance of your custom filter and ensures that your filter does not go out-of-scope while it is being used. This means that you do not have to retain a reference to your filter throughout its lifetime, the CustomFilter class handles this detail for you.

```
// Create an instance of the ExampleWriterContentFilter and then
// create a CustomFilter object which holds a shared pointer to your
// ContentFilter.
std::string my_filter_name = "my_custom_filter";
rti::topic::CustomFilter<ExampleWriterContentFilter>
    my_custom_filter(new ExampleWriterContentFilter());
dds::domain::DomainParticipant participant1(0);
dds::domain::DomainParticipant participant2(0);
// To make sure that we don't miss any samples in this example,
// setup there DataWriter and DataReaders to be reliable
dds::pub::qos::DataWriterQos writer_qos;
writer_qos << dds::core::policy::Reliability::Reliable();
writer_qos << dds::core::policy::Durability::TransientLocal();
writer_qos << dds::core::policy::History::KeepAll();
dds::sub::qos::DataReaderQos reader_qos;
reader_qos << dds::core::policy::Reliability::Reliable();
reader_qos << dds::core::policy::Durability::TransientLocal();
reader_qos << dds::core::policy::History::KeepAll();
```

6.96.3.11 Registering the Filter

After wrapping your content filter with the CustomFilter class, and before the custom filter can be used, it must be registered with RTI Connext:

```
// Register the filter with both participants. Doing this will
// automatically enable writer-side filtering if any DataWriters match with
// any DataReaders that are using the same filter
participant1.extensions().register_contentfilter(my_custom_filter, my_filter_name);
participant2.extensions().register_contentfilter(my_custom_filter, my_filter_name);
```

6.96.3.12 Unregistering the Filter

When the filter is no longer needed, it can be unregistered from RTI Connext:

```
// You do not have to unregister your content filter unless you want to
// register a different filter with the same name
participant1.extensions().unregister_contentfilter(my_filter_name);
participant2.extensions().unregister_contentfilter(my_filter_name);
```

6.96.3.13 Using a CustomFilter

After the custom filter has been registered, you must create **dds::topic::Filter** (p. 1558) that contain the filter expressions and parameters that will be used in your filter. You must give these Filters names that match the name with which the CustomFilter was registered in order to associate the two.

Next, create the readers that will be using the custom filter with dds::topic::ContentFilteredTopics that have been created with the Filters you created.

After that, everything is set up for you. Now, your readers will only receive samples matching the filter that you have set up for them.

```
dds::topic::Topic<Foo> topic1(participant1, "ExampleTopic");
dds::topic::Topic<Foo> topic2(participant2, "ExampleTopic");
// Create the parameter lists for the filters
std::vector<std::string> cft_parameters(2);
// In this example, the first parameter dictates whether or not the
```

```
// writer-side filter optimization will be turned on. Parameter 2
// tells the writer_evaluate which single value of x this reader wants to
// receive samples for
cft_parameters[0] = "1";
cft_parameters[1] = "5";
dds::topic::Filter filter1("x = %1", cft_parameters);
cft_parameters[0] = "1";
cft_parameters[1] = "10";
dds::topic::Filter filter2("x = %1", cft_parameters);
// Any reader that is not using the writer-side filter optimization will
// receive samples where x > 7. The second parameter in this case is
// ignored, but is set here just as a reminder
cft_parameters[0] = "0";
cft_parameters[1] = "7";
dds::topic::Filter filter3("x > %1", cft_parameters);
// Create a DataWriter that will perform writer-side filtering
dds::pub::DataWriter<Foo> writer(
   dds::pub::Publisher(participant1), topic1, writer_qos);
// Assign filters a name or else the default, rti::topic::sql_filter_name,
// will be used
filter1.extensions().name(my_filter_name);
filter2.extensions().name(my_filter_name);
filter3.extensions().name(my_filter_name);
// Create the ContentFilteredTopics that will be used when creating the
// DataReaders
dds::topic::ContentFilteredTopic<Foo> cft1(
   topic2, "MyContentFilteredTopic1", filter1);
dds::topic::ContentFilteredTopic<Foo> cft2(
   topic2, "MyContentFilteredTopic2", filter2);
dds::topic::ContentFilteredTopic<Foo> cft3(
   topic2, "MyContentFilteredTopic3", filter3);
// Create two readers that will make use of the writer-side filter
// optimization and one that won't
dds::sub::DataReader<Foo> optimizedReader1(
   dds::sub::Subscriber(participant2), cft1, reader_qos);
dds::sub::DataReader<Foo> optimizedReader2(
   dds::sub::Subscriber(participant2), cft2, reader_qos);
dds::sub::DataReader<Foo> reader3(
   dds::sub::Subscriber(participant2), cft3, reader_qos);
// Wait for the writer to match all readers before writing any samples
dds::core::cond::StatusCondition status_condition(writer);
int32_t matched_publications = 0;
while (matched_publications != 3) {
    dds::core::status::PublicationMatchedStatus status =
        writer.publication_matched_status();
    matched_publications = status.total_count();
}
// Write 11 samples with x = 0 through 10
for (int i = 0; i < 11; i++) {
    writer.write(Foo(i, i));
}
// optimizedReader1 will only receive samples with x == 5
std::cout « "optimizedReader1's samples: " « std::endl;
dds::sub::LoanedSamples<Foo> samples = optimizedReader1.take();
std::copy(
    samples.begin(),
    samples.end(),
    dds::sub::LoanedSamples<Foo>::ostream_iterator(std::cout, "\n"));
// optimizedReader2 will only receive samples with x == 10
std::cout « "optimizedReader2's samples: " « std::endl;
samples = optimizedReader2.take();
std::copy(
    samples.begin(),
    samples.end(),
    dds::sub::LoanedSamples<Foo>::ostream_iterator(std::cout, "\n"));
// reader3 will receive samples with x == 8, 9, 10
std::cout « "reader3's samples: " « std::endl;
samples = reader3.take();
std::copy(
    samples.begin(),
    samples.end(),
    dds::sub::LoanedSamples<Foo>::ostream_iterator(std::cout, "\n"));
```

6.96.3.14 Looking up the Filter

A custom filter that is registered with a DomainParticipant can be looked up:

```
rti::topic::CustomFilter<ExampleWriterContentFilter> retrieved_custom_filter =
       rti::topic::find_content_filter<ExampleWriterContentFilter>(
           participant1, my_filter_name);
// You can retrieve your ContentFilter from the CustomFilter:
ExampleWriterContentFilter* retrieved_content_filter =
          retrieved_custom_filter.get();
```

6.97 XML Application Creation

Defining DDS systems in XML.

Defining DDS systems in XML.

6.97.1 Introduction

XML-Based Application Creation is a mechanism to simplify the development and programming of RTI Connext applications. RTI Connext supports the use of XML for the complete system definition. This includes not only the definition of the data types and Quality of Service settings, but also the definition of the Topics, DomainParticipants, and all the Entities they contain (Publishers, Subscribers, DataWriters and DataReaders).

The application code simply indicates the participant configuration name of the DomainParticipant that the application wants to create. The XML-Based Application Creation infrastructure takes care of the rest: creating the DomainParticipant, registering the types and Topics, and populating all the configured Entities. When the application needs to read or write data, register listeners, or perform any other action, it simply looks up the appropriate Entity by name and uses it.

See the RTI_ConnextDDS_CoreLibraries_XML_AppCreation_GettingStarted.pdf for a more exhaustive description XML Application Creation, most notably, how to set up the XML Configuration files.

The example on this page shows a basic use of the APIs that are necessary in order to access the entities that have been created in an XML file.

Relevant functions that enable the use of XML Application Creation are:

- **dds::core::QosProvider::create_participant_from_config** (p. 2046)
- **rti::domain::find_participant_by_name** (p. 731)
- **rti::sub::find_subscriber(const dds::domain::DomainParticipant participant,const std::string& subscriber_name)** (p. 758);
- **rti::sub::find_datareader_by_name(dds::sub::Subscriber subscriber, const std::string& datareader_name)** (p. 762)
- **rti::sub::find_datareader_by_name(dds::domain::DomainParticipant participant, const std::string& datareader_name)** (p. 763)
- **rti::pub::find_publisher(const dds::domain::DomainParticipant participant,const std::string& publisher_name)** (p. 744);
- **rti::pub::find_datawriter_by_name(dds::pub::Publisher publisher, const std::string& datawriter_name)** (p. 748)
- **rti::pub::find_datawriter_by_name(dds::domain::DomainParticipant participant, const std::string& datawriter_name)** (p. 749)

6.97.2 Setting up this Example

The following #includes are needed for the examples on this page

```
#include <iostream>
#include <dds/core/QosProvider.hpp>
#include <dds/sub/DataReader.hpp>
#include <dds/sub/find.hpp>
#include <dds/pub/DataWriter.hpp>
#include <dds/pub/find.hpp>
#include "Foo.hpp"
```

The following is the configuration that we will be using in this example:

```
<dds>
    <types>
         <struct name="Foo">
             <member name="x" type="long"/>
          </struct>
    </types>
    <domain_library name="ExampleDomainLibrary" >
        <domain name="ExampleDomain" domain_id="0">
         <register_type name="Foo" kind ="userGenerated"/>
         <topic name="ExampleTopic" register_type_ref="Foo"/>
        </domain>
    </domain_library>
    <participant_library name="ExampleParticipantLibrary">
        <domain_participant name="ExamplePublicationParticipant"
       domain_ref="ExampleDomainLibrary::ExampleDomain">
            <publisher name="ExamplePublisher">
                <data_writer name="ExampleWriter" topic_ref="ExampleTopic"/>
            </publisher>
        </domain_participant>
         <domain_participant name="ExampleSubscriptionParticipant"
       domain_ref="ExampleDomainLibrary::ExampleDomain">
             <data_reader name="ExampleReader" topic_ref="ExampleTopic">
                <content_filter name="ExampleTopic" kind="builtin.sql">
                    <expression> foo > 5 </expression>
                </content_filter>
            </data_reader>
        </domain_participant>
    </participant_library>
</dds>
```

6.97.2.1 Using XML Application Creation

You must first make sure that your configuration file is loaded by RTI Connext by using the QosProvider:

```
    rti::core::QosProviderParams provider_params;
    // Configure the default QosProvider to load the configuration
    // config_file == "/the/path/to/your/xml/configuration.xml"
    provider_params.url_profile({ config_file });
    rti::core::default_qos_provider_params(provider_params);
```

Then, if you are using a user-generated type, you must register the type with RTI Connext:

```
    // When using user-generated types, you must register the type with RTI
    // Connext DDS before creating the participants and the rest of the entities
    // in your system
    rti::domain::register_type<Foo>("Foo");
```

To create and access the system that you have defined in your system, call **dds::core::QosProvider::create_↩ participant_from_config** (p. 2046):

```
    // Create the participants, changing the domain id from the one in the
    // configuration
    rti::domain::DomainParticipantConfigParams params(10);
    // Create the participants
    auto default_provider = dds::core::QosProvider::Default();
    dds::domain::DomainParticipant publication_participant =
            default_provider.extensions().create_participant_from_config(
                    "ExampleParticipantLibrary::ExamplePublicationParticipant",
                    params);
    dds::domain::DomainParticipant subscription_participant =
            default_provider.extensions().create_participant_from_config(
                    "ExampleParticipantLibrary::ExampleSubscriptionParticipant",
```

```
                params);
```

After you have created the participants which were defined in your configuration, you can use various find functions to access the other entities that you have configured. For example, to lookup the DataWriter and DataReader from the example configuration:

```
// Lookup the DataWriter and DataReader from the configuration
dds::pub::DataWriter<Foo> found_writer =
        rti::pub::find_datawriter_by_name<dds::pub::DataWriter<Foo»(
                publication_participant,
                "ExamplePublisher::ExampleWriter");
// The implicit subscriber was used to create this reader so we only
// provide the reader's name, and not a fully qualified name as we did to
// look up the DataWriter
dds::sub::DataReader<Foo> found_reader =
        rti::sub::find_datareader_by_name<dds::sub::DataReader<Foo»(
                subscription_participant,
                "ExampleReader");
```

6.98 Request-Reply Examples

Examples on how to use the request-reply API .

Examples on how to use the request-reply API .

Request-Reply code examples.

6.98.1 Request-Reply Examples

Requesters and Repliers provide a way to use the Request-Reply communication pattern on top of the DDS entities. An application uses a Requester to send requests to a Replier; another application using a Replier receives a request and can send one or more replies for that request. The Requester that sent the request (and only that one) will receive the reply (or replies).

DDS Types

RTI Connext uses DDS data types for sending and receiving requests and replies. Valid types are those generated by the rtiddsgen code generator, the DDS built-in types, and `DynamicData`. Refer to the `Core Libraries User's Manual` and the following links for more information:

- `Code Generator User's Manual`,
- **Using the DDS built-in types** (p. 50),
- **Using DynamicData** (p. 439)

Set up

- **Create a DomainParticipant** (p. 307)
- **Create a requester** (p. 345)
- **Create a requester with parameters** (p. 345)

- **Create a replier** (p. 347)

Requester: sending requests and receiving replies

- **Basic Requester example** (p. 346)
- **Correlation between requests and replies** (p. 346)

Replier: receiving requests and sending replies

- **Basic Replier example** (p. 348)
- **SimpleReplier example** (p. 348)

Note

To use Request-Reply you need to build and link your application with the additional `rticonnextmsgcpp2` library.

6.98.2 Requester Creation

- **Setting up a DomainParticipant** (p. 307)
- Creating a Requester

```
using namespace rti::request;
dds::domain::DomainParticipant participant(domain_id);
Requester<Foo, Bar> requester(participant, "TestService");
```

6.98.3 Creating a Requester with optional parameters

- **Setting up a DomainParticipant** (p. 307)
- Creating a Requester with additional parameters

```
using namespace rti::request;
// Create a DomainParticipant
dds::domain::DomainParticipant participant(domain_id);
// Create a Requester with a QoS profile (located for example in
// USER_QOS_PROFILES.xml, in the current working directory, which the
// default QosProvider will load)
RequesterParams requester_params(participant);
requester_params.service_name("TestService");
dds::core::QosProvider qos_provider = dds::core::QosProvider::Default();
requester_params.datareader_qos(
        qos_provider.datareader_qos("RequestReplyExampleProfiles::RequesterExampleProfile"));
requester_params.datawriter_qos(
        qos_provider.datawriter_qos("RequestReplyExampleProfiles::RequesterExampleProfile"));
Requester<Foo, Bar> requester(requester_params);
```

See also

Requester Creation (p. 345)

Configuring Request-Reply QoS profiles (p. 348)

6.98.4 Basic Requester example

- **Requester Creation** (p. 345)
- **Creating a Requester with optional parameters** (p. 345)
- Basic Requester example

```
using namespace rti::request;
// Send request
Foo request("A Request");
requester.send_request(request);
// Receive reply (wait for it and get the sample)
auto replies = requester.receive_replies(MAX_WAIT);
if (replies.length() == 0) {
    std::cout « "Reply not received\n";
}
for (const auto& reply : replies) {
    if (reply.info().valid()) {
        std::cout « "Received reply: " « reply.data() « std::endl;
    } else {
        std::cout « "Received invalid reply\n";
    }
}
```

See also

Basic Replier example (p. 348)
SimpleReplier example (p. 348)

6.98.5 Correlating requests and replies

- **Requester Creation** (p. 345)
- Example 1) Waiting for a reply to a specific request

```
using namespace rti::request;
// Create requests
Foo request1("Request 1"), request2("Request 2");
// Send requests and save request ID
rti::core::SampleIdentity request_id1 = requester.send_request(request1);
rti::core::SampleIdentity request_id2 = requester.send_request(request2);
// Wait for a reply to the second request
bool received = requester.wait_for_replies(1, MAX_WAIT, request_id2);
if (!received) {
    std::cout « "Did not receive reply for request 2" « std::endl;
    return;
}
// Take that reply
dds::sub::LoanedSamples<Bar> replies = requester.take_replies(request_id2);
if (replies.length() == 0) {
    throw std::runtime_error("did not receive any replies");
}
// This postcondition should always be true
if (replies[0].info()->related_original_publication_virtual_sample_identity()
        != request_id2) {
    throw std::runtime_error("postcondition failed");
}
if (replies[0].info().valid()) {
    std::cout « "Received reply for request 2: "
              « replies[0].data() « std::endl;
}
// Wait for a reply to the first request
received = requester.wait_for_replies(1, MAX_WAIT, request_id1);
if (!received) {
    std::cout « "Did not receive reply for request 1" « std::endl;
    return;
}
// Take that reply
replies = requester.take_replies(request_id1);
if (replies.length() == 0) {
    throw std::runtime_error("did not receive any replies");
}
// This postcondition should always be true
```

```
if (replies[0].info()->related_original_publication_virtual_sample_identity()
        != request_id1) {
    throw std::runtime_error("postcondition failed");
}
if (replies[0].info().valid()) {
    std::cout « "Received reply for request 1: "
              « replies[0].data() « std::endl;
}
```

- Example 2) Correlating a reply after receiving it

```
using namespace rti::request;
// Create requests
Foo request1("Request 1"), request2("Request 2");
// Send requests and save request ID
rti::core::SampleIdentity request_id1 = requester.send_request(request1);
rti::core::SampleIdentity request_id2 = requester.send_request(request2);
// Wait for two replies. In this case we don't mind the
// reception order
bool received = requester.wait_for_replies(2, MAX_WAIT);
if (!received) {
    std::cout « "Replies not received" « std::endl;
    return;
}
// Get all the replies
auto replies = requester.take_replies();
// Find the reply for request 1.
//
// We search using the STL find_if algorithm. The search range is all
// the elements in the replies container (from beginning to end).
// We use a predicate included in the API that evaluates to
// true when a Sample's related_identity equals the identity that
// is passed to the constructor (request_id1, in this case)
auto it = std::find_if(
        replies.begin(),
        replies.end(),
        IsReplyRelatedPredicate<Bar>(request_id1));
if (it != replies.end()) {
    std::cout « "Received reply for request 1: "
              « it->data() « std::endl;
}
// Find the reply for request 2
it = std::find_if(
        replies.begin(),
        replies.end(),
        IsReplyRelatedPredicate<Bar>(request_id2));
if (it != replies.end()) {
    std::cout « "Received reply for request 2: "
              « it->data() « std::endl;
}
```

See also

Basic Requester example (p. 346)

Basic Replier example (p. 348)

6.98.6 Creating a Replier

- **Setting up a DomainParticipant** (p. 307)
- Create a Replier

```
using namespace rti::request;
dds::domain::DomainParticipant participant(domain_id);
// Use the same service name as the Requester's
Replier<Foo, Bar> replier(participant, "TestService");
```

6.98.7 Basic Replier example

- **Creating a Replier** (p. 347)
- Basic Replier example

```
using namespace rti::request;
// Receive requests
dds::sub::LoanedSamples<Foo> requests = replier.receive_requests(MAX_WAIT);
for (const auto& request : requests) {
    if (!request.info().valid()) {
        continue;
    }
    Bar reply("Reply for " + request.data().message());
    // Send a reply for this request, identified by the ID in
    // request.info()
    replier.send_reply(reply, request.info());
    // Note: a replier can send more than one reply for the same request
    // replier.send_reply(Bar("Another reply"), request.info());
}
```

See also

Basic Requester example (p. 346)

6.98.8 SimpleReplier example

```
using namespace rti::request;
SimpleReplier<Foo, Bar> replier(
    participant,
    "TestService",
    [](const Foo& request)
    {
        return Bar(std::string("Simple reply for ") + request.message());
    }
);
// After creation the SimpleReplier is already active and the functor will
// be called upon receiving a request.
```

See also

Basic Requester example (p. 346)

6.98.9 Configuring Request-Reply QoS profiles

If you do not specify your own QoS parameters (in RequesterParams and ReplierParams), a **rti::request::Requester** (p. 2187) and **rti::request::Replier** (p. 2168) are created using a default configuration defined in the built-in profile **rti::core::builtin_profiles::qos_snippet_lib::pattern_rpc()** (p. 480). You use this built-in profile as the base for your own profiles:

```
<?xml version="1.0"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="http://community.rti.com/schema/current/rti_dds_profiles.xsd">
    <qos_library name="RequestReplyExampleProfiles">
        <!-- Use these profiles to customize the Requester or Replier QoS -->
        <!-- These profiles inherit from the built-in profile
             BuiltinQosLib::Pattern.RPC, which defines the default values for
             the DataWriters and DataReaders created by a Requester or Replier
        -->
        <qos_profile name="RequesterExampleProfile" base_name="BuiltinQosLib::Pattern.RPC">
            <!-- Set the QoS for the DataReader created by the Requester -->
            <!--
            <datareader_qos>
                ...
            </datareader_qos>
            -->
```

```
            <!-- Set the QoS for the DataWriter created by the Requester -->
            <!--
            <datawriter_qos>
                ...
            </datawriter_qos>
            -->
        </qos_profile>
        <qos_profile name="ReplierExampleProfile" base_name="BuiltinQosLib::Pattern.RPC">
            <!-- Set the QoS for the DataReader created by the Replier -->
            <!--
            <datareader_qos>
                ...
            </datareader_qos>
            -->
            <!-- Set the QoS for the DataWriter created by the Replier -->
            <!--
            <datawriter_qos>
                ...
            </datawriter_qos>
            -->
        </qos_profile>
    </qos_library>
</dds>
```

The example **Creating a Requester with optional parameters** (p. 345) shows how to create a **rti::request::Requester** (p. 2187) using this profile.

See also

Creating a Requester with optional parameters (p. 345)

Configuring QoS Profiles with XML (p. 302)

6.99 Documentation Roadmap

This section contains a roadmap for the new user with pointers on what to read first.

If you are new to RTI Connext, we recommend starting in the following order:

- See the `Getting Started Guide`. This document provides download and installation instructions. It also lays out the core value and concepts behind the product and takes you step-by-step through the creation of a simple example application.
- The `User's Manual` describes the features of the product and how to use them. It is organized around the structure of the DDS APIs and certain common high-level tasks.
- The documentation in the **RTI Connext DDS API Reference** (p. 357) provides an overview of API classes and modules for the DDS data-centric publish-subscribe (DCPS) package from a programmer's perspective. Start by reading the documentation on the main page.
- After reading the high level module documentation, look at the **Publication Example** (p. 304) and **Subscription Example** (p. 305) for step-by-step examples of creating a publication and subscription. These are hyperlinked code snippets to the full API documentation, and provide a good place to begin learning the APIs.
- Next, work through your own application using the example code files generated by rtiddsgen. See the `Code Generator User's Manual`.
- To integrate similar code into your own application and build system, you will likely need to refer to the `Platform Notes`.

6.100 Conventions

Basic concepts required to use the API correctly.

Basic concepts required to use the API correctly.

There are a few basic conventions that you need to understand to use this API correctly:

- The type system, including value types and reference types
- How to use standard and extension APIs
- Error management through exceptions
- C++11 support

6.100.1 Type system

All types in the API are **value types**, **reference types**, or **move-only types**. In this documentation a type is a value type, unless explicitly marked with <<***reference-type***>> *(*p. *351)* or <<***move-only-type***>> *(*p. *353)*. In some cases the type may be marked with <<***value-type***>> *(*p. *350)* for clarity.

6.100.1.1 Value types

<<***value-type***>> *(*p. *350)*Value types implement value semantics.

Note that in this API, types that don't specify their type semantics are value types by default. **IDL-generated types** (p. 606) are also value types.

Value types provide the following functionality:

- Deep-copy copy constructor and assignment operator.
- Move constructor and move-assignment operator <<***C++11***>> *(*p. *353)*.
- Destructor.
- Equality operators.
- Namespace-level **`swap()`** (p. 1460) function.

Those operations are not documented for each type unless they deviate from their usual behavior.

6.100.1.2 Reference types

<<***reference-type***>> *(*p. *351)*Reference types implement *reference semantics*.

In a reference type copy operations, such as copy-construction and copy-assignment
are *shallow*. The reference types are modeled after shared pointers. Similar to pointers, it is important to distinguish between an entity and a reference (or handle) to it. A single entity may have multiple references. Copying a reference does not copy the entity it is referring to—creating additional references from the existing reference(s) is a relatively inexpensive operation.

The lifecycle of references and the entity they are referring to is not the same. In general, the entity lives as long as there is at least one reference to it. When the last reference to the entity ceases to exists, the entity it is referring to is destroyed.

However, there are some exceptions. It is possible—and often convenient—to *retain* an entity even though it has no references, for example to look it up later. An reference can also be explicitly closed, deleting the object it references, regardless of the existence of other references.

An entity is considered to still be *in use* (i.e., retained) if any of the following conditions are met:

- The entity has one or more references.
- The entity has been explicitly retained with the `retain()` member function.
- The entity created another object that is still in use. For example, while a `Topic` exist, its related `Domain↩ Participant` won't be deleted.
- **[DEPRECATED]** The entity has a non-null listener pointer. (Using a raw pointer as a listener is deprecated. The way to assign a listener is using a `std::shared_ptr<`**`Listener`** (p. `1637)>`. When a `shared_ptr` is used, the entity is not retained.)

All reference types inherit from **`dds::core::Reference`** (p. `2149)`. The reference semantics are implemented using a shared count.

Reference types provide the following functionality:

- **Constructors specific to the type**

  Creates a new object and a reference to it. For example:
  ```
  // Create a domain participant and one reference to it.
  dds::domain::DomainParticipant participant(MY_DOMAIN_ID);
  ```
- **Copy constructor**

  Creates a new reference to an existing object, increasing the reference count. For example:
  ```
  dds::domain::DomainParticipant same_participant = participant;
  ```
- **Assignment operator**

  Replaces the object that was referenced to with a new one, possibly decreasing the reference count of the former object and increasing it for the new one.
- **Constructor from `dds::core::null` (**p. **`438`**`)` (or `nullptr` <<***C++11***>> *(*p. ***353***)**)**

  Creates an instance that doesn't reference any object. For example:
  ```
  dds::domain::DomainParticipant null_participant = dds::core::null;
  ```

- **Assignment operator from `dds::core::null` (p. 438) (or `nullptr` <<*C++11*>> (p. *353*))**

  Makes the reference empty, decreasing the reference count.
  ```
  dds::domain::DomainParticipant participant(MY_DOMAIN_ID);
  participant = dds::core::null; // This will destroy the participant object
  ```

- **Equal operator (to another reference)**

  Returns true only if the referenced object is the same. For example:
  ```
  dds::domain::DomainParticipant participant1(MY_DOMAIN_ID);
  dds::domain::DomainParticipant participant2(MY_DOMAIN_ID);
  if (participant1 != participant2) {
      std::cout « "References to different objects\n";
  }
  // Note that this also destroys the former object referenced to by participant2
  participant2 = participant1;
  if (participant1 == participant2) {
      std::cout « "References to the same object\n";
  }
  ```

  The operators <, <=, >, >= are also supported, and compare the underlying pointer.

- **Equal operator to `dds::core::null` (p. 438) (or `nullptr` <<*C++11*>> (p. *353*))**

  Returns true only if this reference is empty. For example:
  ```
  dds::domain::DomainParticipant participant = dds::core::null;
   // dds::core::null and nullptr are interchangeable in C++11
  if (participant == nullptr) {
      std::cout « "Empty reference\n";
  }
  ```

Some reference types, including **dds::core::Entity** (p. 1520) and its subclasses, also provide:

- **`close()` (p. 1024)**

  Forces the destruction of the underlying object. After this, calling a method on the destroyed object throws **`dds::core::AlreadyClosedError`** (p. 806). For example:
  ```
  dds::domain::DomainParticipant participant1(MY_DOMAIN_ID);
  dds::domain::DomainParticipant participant2 = participant1;
  participant2.close(); // Destroys the underlying object
  int id = participant1.domain_id(); // throws dds::core::AlreadyClosedError
  ```

- **`retain()`**

  Disables the destruction of the underlying object. When all references are destroyed the underlying object still exists and can be looked up. To finally destroy a retained object you need to explicitly call **close()** (p. 1024).

  For example:
  ```
  using namespace dds::domain;
  void create_participant()
  {
      DomainParticipant participant(MY_DOMAIN_ID);
      participant.retain();
  } // participant goes out of scope
  void test_retain()
  {
      create_participant();
      DomainParticipant participant = find(MY_DOMAIN_ID);

      // ...

      // Since the participant has been retained, we need to explicitly close it:
      participant.close();
  }
  ```

Reference types with an inheritance relationship such as **dds::core::Entity** (p. 1520) or **dds::core::cond::Condition** (p. 948) can use **dds::core::polymorphic_cast** (p. 620) to cast from a base to a derived class.

For more examples on reference types see **Entity Use Cases** (p. 325)

6.100.1.3 Move-only types

<<***move-only-type***>> *(p. 353)*Move-only types are types that can't be copied, only "moved." A move-only type encapsulates a view of an internal resource. Only one reference to that resource may exist at a time.

6.100.2 C++11 Support

<<***C++11***>> *(p. 353)*Functionality supported only in C++11.

This API is designed to integrate with and make use of C++11. The API headers are prepared to detect at application-compile time what C++11 features are available and make use of them.

Different compilers support different C++11 features, and some require
explicitly activating C++11 support. If your compiler activates C++11 by default (for example, Visual Studio 2010 and later), you dont need to do anything. Whatever features are available will be used. If your compiler requires an explicit activation, you will need to pass a flag (for example, -std=c++0x or -std=c++11 in gcc and clang). The `Platform Notes` can help with that.

The API provides the following C++11 features when available:

- Defines move constructors and move-assignment operators for most types, both in the API and in types generated from IDL.
- Enables the use of the range for-loop in classes like **`dds::sub::LoanedSamples`** (p. 1663).
- Enables the use of lambda functions in places like **`dds::sub::cond::ReadCondition`** (p. 2133)
- Specifies some key functions as `noexcept`, like move constructors.
- Enables the use of `std::initializer_list` in several functions. For example see **`rti::core::policy::Property`** (p. 1969). <<***extension***>> *(p. 354)*
- Provides utilities to manipulate **`dds::core::xtypes::DynamicType`** (p. 1504) and **`dds::core::xtypes::DynamicData`** (p. 1461) **using tuples** (p. 612). <<***extension***>> *(p. 354)* <<***experimental***>> *(p. 355)*
- Supports the interchangeable use of `std::chrono::duration` and **`dds::core::Duration`** (p. 1448). <<***extension***>> *(p. 354)*
- Supports the interchangeable use of `nullptr` and **`dds::core::null`** (p. 438).

6.100.3 Exceptions in the API

The modern C++ API uses exceptions to report errors.

If a function doesn't document what exceptions it may throw and is not declared `noexcept`, then it may throw any of the **standard exceptions** (p. 427).

Destructors won't throw exceptions. There are cases however where you may need to handle an error during an object destruction. Some classes, like the **reference types** (p. 351) provide a **close()** (p. 1024) operation–which can throw–that destroys the underlying entity.

Note that the API can also throw C++ standard exceptions such as `std::bad_alloc`.

For a few critical operations both a regular exception-throwing function and a noexcept function are provided. The noexcept versions always return an **rti::core::Result** (p. 2209) object, and are always extension functions that need to be accessed with the -> operator. For example, a **dds::sub::DataReader** (p. 976) provides both **`take()`** (p. 1024) and **`take_noexcept()`** (p. 1018):

```
auto samples = reader.take(); // may throw
auto result = reader->take_noexcept(); // never throws
if (result.is_ok()) {
     auto& samples = result.get();
     // ...
}
```

See also

Exceptions (p. 427)

6.100.4 Extensions to the standard API

*<<**extension**>> (p. 354)*The stereotype *<<**extension**>> (p. 354)* indicates that a type or a function is RTI Connext product extension to the standard DDS specification.

The RTI extension APIs complement the standard APIs specified by the OMG DDS specification.

There are the following kinds of extensions: extension methods for a standard class, extension types, and extension standalone functions.

To call an extension methods for an standard class use the **`extensions()`** method or the overloaded **arrow operator (->)**. For example:

```
// Standard class (in dds namespace)
dds::domain::DomainParticipant participant(MY_DOMAIN_ID);
// Call a standard method
participant.assert_liveliness();
// Call an extension method:
participant.extensions().register_durable_subscription(...);
// or:
participant->register_durable_subscription(...);
```

Note that the arrow operator is `noexcept` but the `extensions()` method of a reference type is not (it may throw **dds::core::NullReferenceError** (p. 1870) if the object is **dds::core::null** (p. 438)).

In this documentation extension members in a standard type appear as members of that type, although they are members of a delegate type called through the standard type (calling `extensions()` or the arrow operator). We omit this implementation detail from the API documentation for simplicity.

Extension types reside in the `rti` namespace instead of the `dds` namespace. For example:

```
// FlowController is an extension class and it resides in the rti namespace
rti::pub::FlowController flow_controller(participant);
// All methods in an extension class are called using the usual dot operator
flow_controller.trigger_flow();
```

The following example combines an extension function for a standard type (`Reliability`) and an extension type (`AcknowledgmentKind`):

```
// Standard class (in dds namespace)
dds::core::policy::Reliability reliability;
// Call a standard method
reliability.kind(dds::core::policy::ReliabilityKind::RELIABLE);
// Call an extension method, passing an extension enumeration:
reliabilty.extensions().acknowledgment_kind(
        rti::core::policy::AcknowledgmentKind::APPLICATION_AUTO);
// or:
reliabilty->acknowledgment_kind(
        rti::core::policy::AcknowledgmentKind::APPLICATION_AUTO);
```

Extension standalone functions are also in the `rti` namespace. For example:

```
// Standard function (in the dds namespace)
dds::sub::find<dds::sub::DataReader<Foo> >(
    subscriber, "Foo Topic", std::back_inserter(readers));

// Extension function (in the rti namespace)
rti::sub::find_datareaders(subscriber, std::back_inserter(readers));
```

6.100.5 Experimental

<<***experimental***>> *(*p. *355)*Experimental features subject to change.

- RTI Connext experimental features are used to evaluate new features and get user feedback.
- These features are not guaranteed to be fully supported and might be implemented only for some of the programming languages supported by RTI Connext
- Experimental features may or may not appear in future product releases.
- Experimental features should not be used in production.

6.100.6 Method Parameters

Some times the parameter may have one of the following stereotypes, but in most cases the function signature reveals if the parameter is an input parameter (by value or const-reference), or output parameter (non-const reference).

- <<***in***>> *(*p. *355)*
  - An *input* parameter.
- <<***out***>> *(*p. *355)*
  - An *output* parameter.
- <<***inout***>> *(*p. *355)*
  - An *input* and *output* parameter.

6.101 Namespaces and headers

This section describes the headers and namespaces in the modern C++ API.

6.101.1 Header Files

The modern C++ API is organized using a set of standard header files that you will find in the $NDDSHOME/include/hpp/dds directory of your installation. The headers in this directory are organized into 5 modules defined by the DDS v1.2 Platform Independent Mapping (PIM):

- **domain:** DomainParticipant specific headers
- **pub:** Publisher, DataWriter and other headers specific to the publication of data
- **sub:** Subscriber, DataReader and other headers specific to the consumption of data
- **topic:** Topic management specific headers

- **core:** headers that define classes and DDS types that are used by all of the other modules

You will also find an $NDDSHOME/include/hpp/rti directory in your installation. This directory contains RTI Connext implementation specific details along with **extensions** (p. 354) to the standard API.

To import the DDS API into your application you must include these header files. There are several ways to do this:

1. Include the entire standard API:
```
#include <dds/dds.hpp>
```
2. Include the standard API plus all the extensions:
```
#include <rti/rti.hpp>
```
3. Include complete namespaces, for example:
```
#include <dds/domain/ddsdomain.hpp>
#include <dds/sub/ddssub.hpp>
```
4. Include individual headers, for example:
```
#include <dds/domain/DomainParticipant.hpp>
#include <dds/sub/DataReader.hpp>
```

All of the **Programming How-To** (p. 362) pages start with a section showing which header that the code snippets on that page require.

6.101.2 Namespaces

There are a number of different namespaces in the modern C++ API that organize the API into logical groups. The two main namespaces are `dds` and `rti`.

The `dds` namespace includes all of the standard types, classes and methods. The `rti` namespace includes all RTI **extensions** (p. 354) to the standard API.

The `dds` and `rti` namespaces both contain several nested namespaces. There is a namespace for each of the 5 modules: **dds::core** (p. 616), **dds::domain** (p. 634), **dds::topic** (p. 688), **dds::pub** (p. 645), **dds::sub** (p. 658) (and the corresponding **rti::core** (p. 701), **rti::domain** (p. 727), **rti::topic** (p. 771), **rti::pub** (p. 736), **rti::sub** (p. 751)).

The rti namespace also contains **rti::util** (p. 777) and **rti::config** (p. 699).

There are further specialized namespaces within each module namespace.

All of the modules, excluding core, contain a `qos` namespace that contains the Qos classes for the entities contained in that module:

- **`dds::domain::qos`** (p. 640)
  - `DomainParticipantQos`
  - `DomainParticipantFactoryQos`
- **`dds::pub::qos`** (p. 653)

  - PublisherQos
  - DataWriterQos
- **dds::sub::qos** (p. 681)
  - SubscriberQos
  - DataReaderQos
- **dds::topic::qos** (p. 695)
  - TopicQos

All of the individual Qos policies can be accessed from either the **dds::core::policy** (p. 627) namespace for standard Qos policies or the rti::core::policy namespace for RTI extension Qos policies.

In addition to the policy namespace, the core namespace contains two other nested namespaces.

- **dds::core::status** (p. 630)
  - Contains all communication statuses and StatusMask
- **dds::core::cond** (p. 627)
  - Condition
  - GuardCondidtion
  - StatusCondition
  - Waitset

The sub namespace also contains the status and cond nested namespaces.

- **dds::sub::status** (p. 686)
  - DataState
- **dds::sub::cond** (p. 680)
  - QueryCondition
  - ReadCondition

Two convenience namespaces bring all the symbols into a single place: **dds::all** (p. 615) and **rti::all** (p. 699).

6.102 RTI Connext DDS API Reference

RTI Connext modules following the DDS module definitions.

Modules

- **Domain Module**

  *Contains the **dds::domain::DomainParticipant** (p. 1326) class that acts as an entrypoint of RTI Connext and acts as a factory for many of the classes. The **dds::domain::DomainParticipant** (p. 1326) also acts as a container for the other objects that make up RTI Connext.*
- **Topic Module**

  *Contains the **dds::topic::Topic** (p. 2474), **dds::topic::ContentFilteredTopic** (p. 954), and MultiTopic classes, the TopicListener interface, and more generally, all that is needed by an application to define **dds::topic::Topic** (p. 2474) objects and attach QoS policies to them.*
- **Publication Module**

  *Contains the **rti::pub::FlowController** (p. 1572), **dds::pub::Publisher** (p. 1993), and **dds::pub::DataWriter** (p. 1138) classes as well as the PublisherListener and **dds::pub::DataWriterListener** (p. 1202) interfaces, and more generally, all that is needed on the publication side.*
- **Subscription Module**

  *Contains the **dds::sub::Subscriber** (p. 2410), **dds::sub::DataReader** (p. 976), **dds::sub::cond::ReadCondition** (p. 2133), **dds::sub::cond::QueryCondition** (p. 2059), and **rti::sub::TopicQuery** (p. 2516) classes, as well as the **dds::sub::SubscriberListener** (p. 2422) and **dds::sub::DataReaderListener** (p. 1055) interfaces, and more generally, all that is needed on the subscription side.*
- **Infrastructure Module**

  Defines the abstract classes and the interfaces that are refined by the other modules. Contains common definitions such as return codes, status values, and QoS policies.
- **Transports**

  APIs related to RTI Connext pluggable transports.
- **Queries and Filters Syntax**
- **Logging and Version**

  APIs of troubleshooting utilities and APIs designed to configure the overall behavior of RTI Connext.
- **General Utilities and Compliance Configuration**

  API of general utilities used in the RTI Connext distribution.
- **Observability Framework**

  RTI Connext Observability Framework modules and APIs.
- **Durability and Persistence**

  APIs related to RTI Connext Durability and Persistence.
- **System Properties**

  System Properties.
- **Configuring QoS Profiles with XML**

  APIs related to XML QoS Profiles.
- **Additional RTI Connext Communication Patterns**

  Extensions to the RTI Connext publish-subscribe functionality.

6.102.1 Detailed Description

RTI Connext modules following the DDS module definitions.

6.102.2 Overview

Information flows with the aid of the following constructs: **dds::pub::Publisher** (p. 1993) and **dds::pub::DataWriter** (p. 1138) on the sending side, **dds::sub::Subscriber** (p. 2410) and **dds::sub::DataReader** (p. 976) on the receiving side.

- A **dds::pub::Publisher** (p. 1993) is an object responsible for data distribution. It may publish data of different data types. A TDataWriter acts as a *typed* (i.e. each **dds::pub::DataWriter** (p. 1138) object is dedicated to one application data type) accessor to a publisher. A **dds::pub::DataWriter** (p. 1138) is the object the application must use to communicate to a publisher the existence and value of data objects of a given type. When data object values have been communicated to the publisher through the appropriate data-writer, it is the publisher's responsibility to perform the distribution (the publisher will do this according to its own QoS, or the QoS attached to the corresponding data-writer). A *publication* is defined by the association of a data-writer to a publisher. This association expresses the intent of the application to publish the data described by the data-writer in the context provided by the publisher.
- A **dds::sub::Subscriber** (p. 2410) is an object responsible for receiving published data and making it available (according to the Subscriber's QoS) to the receiving application. It may receive and dispatch data of different specified types. To access the received data, the application must use a *typed* TDataReader attached to the subscriber. Thus, a *subscription* is defined by the association of a data-reader with a subscriber. This association expresses the intent of the application to subscribe to the data described by the data-reader in the context provided by the subscriber.

dds::topic::Topic (p. 2474) objects conceptually fit between publications and subscriptions. Publications must be known in such a way that subscriptions can refer to them unambiguously. A **dds::topic::Topic** (p. 2474) is meant to fulfill that purpose: it associates a name (unique in the domain i.e. the set of applications that are communicating with each other), a data type, and QoS related to the data itself. In addition to the topic QoS, the QoS of the **dds::pub::↩ DataWriter** (p. 1138) associated with that Topic and the QoS of the **dds::pub::Publisher** (p. 1993) associated to the **dds::pub::DataWriter** (p. 1138) control the behavior on the publisher's side, while the corresponding **dds::topic::↩ Topic** (p. 2474), **dds::sub::DataReader** (p. 976) and **dds::sub::Subscriber** (p. 2410) QoS control the behavior on the subscriber's side.

When an application wishes to publish data of a given type, it must create a **dds::pub::Publisher** (p. 1993) (or reuse an already created one) and a **dds::pub::DataWriter** (p. 1138) with all the characteristics of the desired publication. Similarly, when an application wishes to receive data, it must create a **dds::sub::Subscriber** (p. 2410) (or reuse an already created one) and a **dds::sub::DataReader** (p. 976) to define the subscription.

6.102.3 Conceptual Model

The overall conceptual model is shown below.

Notice that all the main communication objects (the specializations of Entity) follow unified patterns of:

- Supporting QoS (made up of several QosPolicy); QoS provides a generic mechanism for the application to control the behavior of the Service and tailor it to its needs. Each **dds::core::Entity** (p. 1520) supports its own specialized kind of QoS policies (see **All QoS Policies** (p. 508)).

- Accepting a **Listener** (p. 1637); listeners provide a generic mechanism for the middleware to notify the application of relevant asynchronous events, such as arrival of data corresponding to a subscription, violation of a QoS setting, etc. Each **dds::core::Entity** (p. 1520) supports its own specialized kind of listener. Listeners are related to changes in status conditions (see **Status Kinds** (p. 429)).

  Note that only one **Listener** (p. 1637) per entity is allowed (instead of a list of them). The reason for that choice is that this allows a much simpler (and, thus, more efficient) implementation as far as the middleware is concerned. Moreover, if it were required, the application could easily implement a listener that, when triggered, triggers in return attached 'sub-listeners'.

- Accepting a **dds::core::cond::StatusCondition** (p. 2372) (and a set of **dds::sub::cond::ReadCondition** (p. 2133) objects for the **dds::sub::DataReader** (p. 976)); conditions (in conjunction with **dds::core::cond::WaitSet** (p. 2617) objects) provide support for an alternate communication style between the middleware and the application (i.e., wait-based rather than notification-based).

All DCPS entities are attached to a **dds::domain::DomainParticipant** (p. 1326). A domain participant represents the local membership of the application in a domain. A *domain* is a distributed concept that links all the applications able to communicate with each other. It represents a communication plane: only the publishers and the subscribers attached to the same domain may interact.

DomainEntity is an intermediate object whose only purpose is to state that a DomainParticipant cannot contain other domain participants.

At the DCPS level, data types represent information that is sent atomically. For performance reasons, only plain data structures are handled by this level.

By default, each data modification is propagated individually, independently, and uncorrelated with other modifications. However, an application may request that several modifications be sent as a whole and interpreted as such at the recipient side. This functionality is offered on a Publisher/Subscriber basis. That is, these relationships can only be specified among **dds::pub::DataWriter** (p. 1138) objects attached to the same **dds::pub::Publisher** (p. 1993) and retrieved among **dds::sub::DataReader** (p. 976) objects attached to the same **dds::sub::Subscriber** (p. 2410).

By definition, a **dds::topic::Topic** (p. 2474) corresponds to a single data type. However, several topics may refer to the same data type. Therefore, a **dds::topic::Topic** (p. 2474) identifies data of a single type, ranging from one single instance to a whole collection of instances of that given type. This is shown below for the hypothetical data type **Foo** (p. 1588).

In case a set of instances is gathered under the same topic, different instances must be distinguishable. This is achieved by means of the values of some data fields that form the **key** to that data set. The *key description* (i.e., the list of data fields whose value forms the key) has to be indicated to the middleware. The rule is simple: *different data samples with the same key value represent successive values for the same instance, while different data samples with different key values represent different instances*. If no key is provided, the data set associated with the **dds::topic::Topic** (p. 2474) is restricted to a *single instance*.

Topics need to be known by the middleware and potentially propagated. Topic objects are created using the create operations provided by **dds::domain::DomainParticipant** (p. 1326).

The interaction style is straightforward on the publisher's side: when the application decides that it wants to make data available for publication, it calls the appropriate operation on the related **dds::pub::DataWriter** (p. 1138) (this, in turn, will trigger its **dds::pub::Publisher** (p. 1993)).

On the subscriber's side however, there are more choices: relevant information may arrive when the application is busy doing something else or when the application is just waiting for that information. Therefore, depending on the way the application is designed, asynchronous notifications or synchronous access may be more appropriate. Both interaction

modes are allowed, a **Listener** (p. 1637) is used to provide a callback for synchronous access and a **dds::core::cond↩ ::WaitSet** (p. 2617) associated with one or several **dds::core::cond::Condition** (p. 948) objects provides asynchronous data access.

The same synchronous and asynchronous interaction modes can also be used to access changes that affect the middleware communication status (see **Status Kinds** (p. 429)). For instance, this may occur when the middleware asynchronously detects an inconsistency. In addition, other middleware information that may be relevant to the application (such as the list of the existing topics) is made available by means of **built-in topics** (p. 46) that the application can access as plain application data, using built-in data-readers.

6.102.4 Modules

DCPS consists of five modules:

- **Infrastructure module** (p. 72) defines the abstract classes and the interfaces that are refined by the other modules. It also provides support for the two interaction styles (notification-based and wait-based) with the middleware.
- **Domain module** (p. 44) contains the **dds::domain::DomainParticipant** (p. 1326) class that acts as an entrypoint of the Service and acts as a factory for many of the classes. The **dds::domain::DomainParticipant** (p. 1326) also acts as a container for the other objects that make up the Service.
- **Topic module** (p. 47) contains the **dds::topic::Topic** (p. 2474) class, the TopicListener interface, and more generally, all that is needed by the application to define **dds::topic::Topic** (p. 2474) objects and attach QoS policies to them.
- **Publication module** (p. 55) contains the **dds::pub::Publisher** (p. 1993) and **dds::pub::DataWriter** (p. 1138) classes as well as the PublisherListener and **dds::pub::DataWriterListener** (p. 1202) interfaces, and more generally, all that is needed on the publication side.
- **Subscription module** (p. 61) contains the **dds::sub::Subscriber** (p. 2410), **dds::sub::DataReader** (p. 976), **dds::sub::cond::ReadCondition** (p. 2133), and **dds::sub::cond::QueryCondition** (p. 2059) classes, as well as the **dds::sub::SubscriberListener** (p. 2422) and **dds::sub::DataReaderListener** (p. 1055) interfaces, and more generally, all that is needed on the subscription side.

6.103 Additional RTI Connext Communication Patterns

Extensions to the RTI Connext publish-subscribe functionality.

Modules

- **Request-Reply Pattern**

  Support for the request-reply communication pattern.
- **Queuing Pattern**

  Support for the queuing communication pattern.
- **Remote Procedure Call**

  Remote Procedure call (RPC) communication pattern.
- **Utilities**

  Utilities for the RTI Connext Messaging module.

6.103.1 Detailed Description

Extensions to the RTI Connext publish-subscribe functionality.

6.104 Programming How-To's

These "How To"s illustrate how to apply RTI Connext APIs to common use cases.

Modules

- **Publication Example**

  A data publication example.
- **Subscription Example**

  A data subscription example.
- **Participant Use Cases**

  Working with domain participants.
- **Topic Use Cases**
- **Publisher Use Cases**
- **DataWriter Use Cases**
- **Subscriber Use Cases**
- **DataReader Use Cases**
- **Entity Use Cases**
- **Waitset Use Cases**

  Using WaitSets and Conditions.
- **Filter Use Cases**

  Working with data filters.
- **Creating Custom Content Filters**

  Creating a custom content filter.
- **XML Application Creation**

  Defining DDS systems in XML.
- **Request-Reply Examples**

  Examples on how to use the request-reply API .
- **RPC Tutorial**

  Getting Started with Remote Procedure Call with DDS.
- **Built-in Types Examples**

  Using Built-in Types.
- **Exceptions**

  How DDS return codes map to C++ exceptions.
- **Qos Use Cases**
- **Qos Provider Use Cases**

  *How to use **dds::core::QosProvider*** (p. 2025) *to access XML QoS profiles.*
- **Working with IDL types**

  How IDL types map to C++ classes.
- **DynamicType and DynamicData Use Cases**

  Using DynamicType and DynamicData.

6.104.1 Detailed Description

These "How To"s illustrate how to apply RTI Connext APIs to common use cases.

These are a good starting point to familiarize yourself with DDS. You can use these code fragments as "templates" for writing your own code.

6.105 Interface

Abstraction of a Transport Plugin network interface.

Classes

- struct **NDDS_Transport_Interface_t**

  Storage for the description of a network interface used by a Transport Plugin.

Enumerations

- enum **NDDS_Transport_Interface_Status_t** {
  NDDS_TRANSPORT_INTERFACE_OFF = 0 ,
  NDDS_TRANSPORT_INTERFACE_ON = 1 }

  Interface status.

6.105.1 Detailed Description

Abstraction of a Transport Plugin network interface.

A Transport Plugin may be able to use several logical or physical network interfaces in a single node (machine). For example, there may be multiple NICs for IP networks or multiple serial ports for serial networks.

An instance of a Transport Plugin is a conduit to one or more network interfaces associated with the transport.

Instances of a Transport Plugin must assign a unique unicast address to each of the network interfaces that it can use to send and receive messages. The unicast address should be within the range of addresses that are addressable by the Transport Plugin (as defined by the plugin itself).

Then, when RTI Connext sends a message to an unicast destination address, the destination address will be made of two parts. The network address and an interface address. The network address portion will be used by RTI Connext to select the Transport Plugin instance that will send the message. The interface address portion is passed to the Transport Plugin instance as the destination interface to which the message should be sent.

See also

NDDS_Transport_Address_t (p. 1784) **NDDS_Transport_ClassId_t** (p. 371)

6.105.2 Enumeration Type Documentation

6.105.2.1 NDDS_Transport_Interface_Status_t

enum **NDDS_Transport_Interface_Status_t**

Interface status.

Enumerator

NDDS_TRANSPORT_INTERFACE_OFF	The transport interface is OFF.
NDDS_TRANSPORT_INTERFACE_ON	The transport interface is ON.

6.106 Transport Plugins Configuration

Transport plugins configuration with RTI Connext.

Classes

- struct **NDDS_Transport_UUID**

  Unequivocally identifies a transport plugin instance.
- struct **TransportAllocationSettings_t**

  Allocation settings used by various internal buffers.
- struct **NDDS_Transport_Property_t**

  Base configuration structure that must be inherited by derived Transport Plugin classes.

Macros

- #define **NDDS_TRANSPORT_PORT_INVALID** ((**NDDS_Transport_Port_t**) 0)

  Port 0 is considered to be invalid.
- #define **NDDS_TRANSPORT_UUID_SIZE** 12

  *Size of a **NDDS_Transport_UUID** (p. 1829).*
- #define **NDDS_TRANSPORT_LENGTH_UNLIMITED** -1

  Represent an unlimited length.
- #define **NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN** 0

  Rank interface as unknown or not yet set.
- #define **NDDS_TRANSPORT_UUID_UNKNOWN**

  Value for UUIDs that have no known value. Used as default.
- #define **NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED** (-1)

  *The constant used as 'unlimited' for the 'max_count' field of the structure **TransportAllocationSettings_t** (p. 2533).*
- #define **NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC** (-1)

  *The constant used as 'automatic' for the 'incremental_count' field of the structure **TransportAllocationSettings_t** (p. 2533).*
- #define **NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT**

  *The constant used as default value for the struct **TransportAllocationSettings_t** (p. 2533).*
- #define **NDDS_TRANSPORT_CLASSID_INVALID** (-1)

  Invalid Transport Class ID.
- #define **NDDS_TRANSPORT_CLASSID_UDPv4** (1)

  Builtin IPv4 UDP/IP Transport Plugin class ID.
- #define **NDDS_TRANSPORT_CLASSID_SHMEM** (0x01000000)

  Builtin Shared-Memory Transport Plugin class ID.

- #define **NDDS_TRANSPORT_CLASSID_SHMEM_510** (2)

  Builtin Shared-Memory Transport Plugin class ID for Connext 5.1.0 and earlier.
- #define **NDDS_TRANSPORT_CLASSID_UDPv6** (2)

  Builtin IPv6 UDP/IP Transport Plugin class ID.
- #define **NDDS_TRANSPORT_CLASSID_UDPv6_510** (5)

  Builtin IPv6 UDP/IP Transport Plugin class ID for Connext 5.1.0 and earlier.
- #define **NDDS_TRANSPORT_CLASSID_TCPV4_LAN** (8)

  IPv4 TCP/IP Transport Plugin class ID for LAN case.
- #define **NDDS_TRANSPORT_CLASSID_TCPV4_WAN** (9)

  IPv4 TCP/IP Transport Plugin class ID for WAN case.
- #define **NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN** "tcpv4_wan"

  IPv4 TCP/IP Transport Plugin class name for WAN case.
- #define **NDDS_TRANSPORT_CLASSID_TLSV4_LAN** (10)

  IPv4 TCP/IP Transport Plugin class ID for LAN case with TLS enabled.
- #define **NDDS_TRANSPORT_CLASSID_TLSV4_WAN** (11)

  IPv4 TCP/IP Transport Plugin class ID for WAN case with TLS enabled.
- #define **NDDS_TRANSPORT_CLASSID_UDPv4_WAN** (0x01000001)

  Builtin IPv4 UDP/IP Asymmetric Transport Plugin class ID.
- #define **NDDS_TRANSPORT_CLASSID_RESERVED_RANGE** (1000)

  Transport Plugin class IDs below this are reserved by RTI.
- #define **NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED** (0x2)

  Specified zero-copy behavior of transport.
- #define **NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN** (3)

  Minimum number of gather-send buffers that must be supported by a Transport Plugin implementation.

Typedefs

- typedef RTI_UINT32 **NDDS_Transport_Port_t**

  Type for storing RTI Connext RTPS ports.
- typedef RTI_INT32 **NDDS_Transport_ClassId_t**

  Type for storing RTI Connext Transport Plugin class IDs.

6.106.1 Detailed Description

Transport plugins configuration with RTI Connext.

Transport plugins are configured using properties. Each transport plugin must derive its property from a base configuration structure **NDDS_Transport_Property_t** (p. 1786).

To see how to configure the Built-in Transport Plugins, see **Built-in Transport Plugins** (p. 83).

See also

Built-in Transport Plugins (p. 83)

6.106.2 Macro Definition Documentation

6.106.2.1 NDDS_TRANSPORT_PORT_INVALID

```
#define NDDS_TRANSPORT_PORT_INVALID (( NDDS_Transport_Port_t) 0)
```

Port 0 is considered to be invalid.

6.106.2.2 NDDS_TRANSPORT_UUID_SIZE

```
#define NDDS_TRANSPORT_UUID_SIZE 12
```

Size of a **NDDS_Transport_UUID** (p. 1829).

6.106.2.3 NDDS_TRANSPORT_LENGTH_UNLIMITED

```
#define NDDS_TRANSPORT_LENGTH_UNLIMITED -1
```

Represent an unlimited length.

6.106.2.4 NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN

```
#define NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN 0
```

Rank interface as unknown or not yet set.

6.106.2.5 NDDS_TRANSPORT_UUID_UNKNOWN

```
#define NDDS_TRANSPORT_UUID_UNKNOWN
```

Value:

```
{ \
  { \
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \
  } \
}
```

Value for UUIDs that have no known value. Used as default.

6.106.2.6 NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED

```
#define NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED (-1)
```

The constant used as 'unlimited' for the 'max_count' field of the structure **TransportAllocationSettings_t** (p. 2533).

6.106.2.7 NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC

```
#define NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC (-1)
```

The constant used as 'automatic' for the 'incremental_count' field of the structure **TransportAllocationSettings_t** (p. 2533).

Automatic means the buffer size will double at every reallocation.

6.106.2.8 NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT

```
#define NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT
```

Value:

```
        {                                   \
        2L, /* initial_count */                                         \
        NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED, /* max_count */     \
        NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC /* incremental_count */ \
}
```

The constant used as default value for the struct **TransportAllocationSettings_t** (p. 2533).

The default value defined in this constant, sets the buffer to have:

- initial_count = 2 elements
- max_count = **NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED** (p. 367)
- incremental_count = NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC

6.106.2.9 NDDS_TRANSPORT_CLASSID_INVALID

```
#define NDDS_TRANSPORT_CLASSID_INVALID (-1)
```

Invalid Transport Class ID.

Transport-Plugins implementations should set their class ID to a value different than this.

6.106.2.10 NDDS_TRANSPORT_CLASSID_UDPv4

```
#define NDDS_TRANSPORT_CLASSID_UDPv4 (1)
```

Builtin IPv4 UDP/IP Transport Plugin class ID.

6.106.2.11 NDDS_TRANSPORT_CLASSID_SHMEM

```
#define NDDS_TRANSPORT_CLASSID_SHMEM (0x01000000)
```

Builtin Shared-Memory Transport Plugin class ID.

6.106.2.12 NDDS_TRANSPORT_CLASSID_SHMEM_510

```
#define NDDS_TRANSPORT_CLASSID_SHMEM_510 (2)
```

Builtin Shared-Memory Transport Plugin class ID for Connext 5.1.0 and earlier.

6.106.2.13 NDDS_TRANSPORT_CLASSID_UDPv6

```
#define NDDS_TRANSPORT_CLASSID_UDPv6 (2)
```

Builtin IPv6 UDP/IP Transport Plugin class ID.

6.106.2.14 NDDS_TRANSPORT_CLASSID_UDPv6_510

```
#define NDDS_TRANSPORT_CLASSID_UDPv6_510 (5)
```

Builtin IPv6 UDP/IP Transport Plugin class ID for Connext 5.1.0 and earlier.

6.106.2.15 NDDS_TRANSPORT_CLASSID_TCPV4_LAN

```
#define NDDS_TRANSPORT_CLASSID_TCPV4_LAN (8)
```

IPv4 TCP/IP Transport Plugin class ID for LAN case.

6.106.2.16 NDDS_TRANSPORT_CLASSID_TCPV4_WAN

```
#define NDDS_TRANSPORT_CLASSID_TCPV4_WAN (9)
```

IPv4 TCP/IP Transport Plugin class ID for WAN case.

6.106.2.17 NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN

```
#define NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN "tcpv4_wan"
```

IPv4 TCP/IP Transport Plugin class name for WAN case.

6.106.2.18 NDDS_TRANSPORT_CLASSID_TLSV4_LAN

```
#define NDDS_TRANSPORT_CLASSID_TLSV4_LAN (10)
```

IPv4 TCP/IP Transport Plugin class ID for LAN case with TLS enabled.

6.106.2.19 NDDS_TRANSPORT_CLASSID_TLSV4_WAN

```
#define NDDS_TRANSPORT_CLASSID_TLSV4_WAN (11)
```

IPv4 TCP/IP Transport Plugin class ID for WAN case with TLS enabled.

6.106.2.20 NDDS_TRANSPORT_CLASSID_UDPv4_WAN

```
#define NDDS_TRANSPORT_CLASSID_UDPv4_WAN (0x01000001)
```

Builtin IPv4 UDP/IP Asymmetric Transport Plugin class ID.

6.106.2.21 NDDS_TRANSPORT_CLASSID_RESERVED_RANGE

```
#define NDDS_TRANSPORT_CLASSID_RESERVED_RANGE (1000)
```

Transport Plugin class IDs below this are reserved by RTI.

User-defined Transport-Plugins should use a class ID greater than this number.

6.106.2.22 NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED

```
#define NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED (0x2)
```

Specified zero-copy behavior of transport.

A Transport Plugin may commit to one of three behaviors for zero copy receives:

1. Always does zero copy.
2. Sometimes does zero copy, up to the transport discretion.
3. Never does zero copy.

This bit should be set only if the Transport Plugin commits to always doing a zero copy receive, or more specifically, always loaning a buffer through its `receive_rEA()` call.

In that case, RTI Connext will not need to allocate storage for a message that it retrieves with the `receive_rEA()` call.

6.106.2.23 NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN

```
#define NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN (3)
```

Minimum number of gather-send buffers that must be supported by a Transport Plugin implementation.

For the **NDDS_Transport_Property_t** (p. 1786) structure to be valid, the value of **NDDS_Transport_Property_t↩ ::gather_send_buffer_count_max** (p. 1789) must be greater than or equal to this value.

6.106.3 Typedef Documentation

6.106.3.1 NDDS_Transport_Port_t

```
typedef RTI_UINT32  NDDS_Transport_Port_t
```

Type for storing RTI Connext RTPS ports.

Unlike IPv4 Socket API ports, which are 2 bytes long, the RTI Connext representation of an RTPS port is 4 bytes.

6.106.3.2 NDDS_Transport_ClassId_t

```
typedef RTI_INT32 NDDS_Transport_ClassId_t
```

Type for storing RTI Connext Transport Plugin class IDs.

Each implementation of a Transport Plugin must have a unique ID. For example, a UDP/IP Transport Plugin implementation must have a different ID than a Shared Memory Transport Plugin.

User-implemented Transport Plugins must have an ID higher than **NDDS_TRANSPORT_CLASSID_RESERVED_RANGE** (p. 370).

6.107 Transport Address

Transport-independent addressing scheme using IPv6 presentation strings and numerically stored in network-ordered format.

Classes

- struct **NDDS_Transport_Address_t**

  Addresses are stored individually as network-ordered bytes.

Macros

- #define **NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER**

  An invalid transport address. Used as an initializer.
- #define **NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE** (72)

  *The minimum size of the buffer that should be passed to **NDDS_Transport_Address_to_string** (p. 374).*

Functions

- RTIBool **NDDS_Transport_Address_to_string** (const **NDDS_Transport_Address_t** *self, char *buffer_inout, size_t buffer_size_in)

  Converts a numerical address to a printable string representation.
- RTIBool **NDDS_Transport_Address_to_string_with_protocol_family_format** (const **NDDS_Transport_Address_t** *me, char *buffer, size_t buffer_size_in, RTIOsapiSocketAFKind family)

  Converts a numerical address to a printable string representation with IPv4 dotted notation or IPv6 presentation string depending on the provided protocol family.
- RTI_INT32 **NDDS_Transport_Address_from_string** (**NDDS_Transport_Address_t** *address_out, const char *address_in)

  Converts an address (IPv4 dotted notation or IPv6 presentation string) into a numerical address.
- void **NDDS_Transport_Address_print** (const **NDDS_Transport_Address_t** *address_in, const char *desc_in, RTI_INT32 indent_in)

  Prints an address to standard out.
- RTI_INT32 **NDDS_Transport_Address_is_ipv4** (const **NDDS_Transport_Address_t** *address_in)

  Checks if an address is an IPv4 address.
- RTI_INT32 **NDDS_Transport_Address_is_multicast** (const **NDDS_Transport_Address_t** *address_in)

  Checks if an address is an IPv4 or IPv6 multicast address.

Variables

- const **NDDS_Transport_Address_t NDDS_TRANSPORT_ADDRESS_INVALID**

  An invalid transport address.

6.107.1 Detailed Description

Transport-independent addressing scheme using IPv6 presentation strings and numerically stored in network-ordered format.

The APIs of RTI Connext uses IPv6 address notation for all transports.

Transport Plugin implementations that are not IP-based are required to map whatever addressing scheme natively used by the physical transport (if any) to an address in IPv6 notation and vice versa.

IPv6 addresses are numerically stored in 16 bytes. An IPv6 address can be presented In string notation in a variety of ways. For example,

```
"00AF:0000:0037:FE01:0000:0000:034B:0089"
"AF:0:37:FE01:0:0:34B:89"
"AF:0:37:FE01::34B:89"
```

are all valid IPv6 presentation of the same address.

IPv4 address in dot notation can be used to specify the last 4 bytes of the address. For example,

```
"0000:0000:0000:0000:0000:0000:192.168.0.1"
"0:0:0:0:0:0:192.168.0.1"
"::192.168.0.1"
```

are all valid IPv6 presentation of the same address.

For a complete description of valid IPv6 address notation, consult the IPv6 Addressing Architecture (RFC 2373).

Addresses are divided into unicast addresses and multicast addresses.

Multicast addresses are defined as

- Addresses that start with 0xFF. That is `FFxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx`.

or an IPv4 multicast address

- Address in the range [::224.0.0.0, ::239.255.255.255]

Multicast addresses do not refer to any specific destination (network interface). Instead, they usually refer to a group of network interfaces, often called a "multicast group".

Unicast addresses always refer to a specific network interface.

6.107.2 Macro Definition Documentation

6.107.2.1 NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER

```
#define NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER
```

Value:

```
{ \
  { \
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \
  } \
}
```

An invalid transport address. Used as an initializer.

For example: **NDDS_Transport_Address_t** (p. 1784) address = NDDS_TRANSPORT_ADDRESS_INVALID_↩ INITIALIZER;

6.107.2.2 NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE

```
#define NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE (72)
```

The minimum size of the buffer that should be passed to **NDDS_Transport_Address_to_string** (p. 374).

For regular addresses, the string size needs to be at least 40 to include space for 8 tuples of 4 characters each plus 7 delimiting colons plus a terminating NULL.

To support UDPv4_WAN strings, it has been adjusted to 72 to fit the following representation (plus NULL terminator): f=XXXXRBPU,u={FF,FF,FF,FF,FF,FF,FF,FF},p=255.255.255.255:65555:65555

6.107.3 Function Documentation

6.107.3.1 NDDS_Transport_Address_to_string()

```
RTIBool NDDS_Transport_Address_to_string (
            const  NDDS_Transport_Address_t * self,
            char * buffer_inout,
            size_t buffer_size_in )
```

Converts a numerical address to a printable string representation.

Precondition

The `buffer_inout` provided must be at least **NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE** (p. 374) characters long.

Parameters

self	<<***in***>> *(p. 355)* The address to be converted.
buffer_inout	<<***inout***>> *(p. 355)* Storage passed in which to return the string corresponding to the address.
buffer_size_in	<<***in***>> *(p. 355)* The size of the storage buffer. Must be >= **NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE** (p. 374)

Returns

1 upon success; 0 upon failure (not enough space in the provided buffer)

6.107.3.2 NDDS_Transport_Address_to_string_with_protocol_family_format()

```
RTIBool NDDS_Transport_Address_to_string_with_protocol_family_format (
            const  NDDS_Transport_Address_t * me,
            char * buffer,
            size_t buffer_size_in,
            RTIOsapiSocketAFKind family )
```

Converts a numerical address to a printable string representation with IPv4 dotted notation or IPv6 presentation string depending on the provided protocol family.

Precondition

The `buffer_inout` provided must be at least **NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE** (p. 374) characters long.

Parameters

me	<<***in***>> *(p. 355)* The address to be converted.
buffer	<<***inout***>> *(p. 355)* Storage passed in which to return the string corresponding to the address.
buffer_size_in	<<***in***>> *(p. 355)* The size of the storage buffer. Must be >= **NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE** (p. 374)
family	<<***in***>> *(p. 355)* The protocol family RTI_OSAPI_SOCKET_AF_INET or RTI_OSAPI_SOCKET_AF_INET6

Returns

RTI_TRUE upon success; 0 upon failure (not enough space in the provided buffer)

6.107.3.3 NDDS_Transport_Address_from_string()

```
RTI_INT32 NDDS_Transport_Address_from_string (
              NDDS_Transport_Address_t * address_out,
             const char * address_in )
```

Converts an address (IPv4 dotted notation or IPv6 presentation string) into a numerical address.

The address string must be in IPv4 dotted notation (X.X.X.X) or IPv6 presentation notation. The string cannot be a hostname since this function does not perform a hostname lookup.

Parameters

address_out	<<***out***>> *(*p. *355)* Numerical value of the address.
address_in	<<***in***>> *(*p. *355)* String representation of an address.

Returns

1 if address_out contains a valid address, 0 if it was not able to convert the string into an address.

6.107.3.4 NDDS_Transport_Address_print()

```
void NDDS_Transport_Address_print (
             const  NDDS_Transport_Address_t * address_in,
             const char * desc_in,
             RTI_INT32 indent_in )
```

Prints an address to standard out.

Parameters

address_in	<<***in***>> *(*p. *355)* Address to be printed.
desc_in	<<***in***>> *(*p. *355)* A prefix to be printed before the address.
indent_in	<<***in***>> *(*p. *355)* Indentation level for the printout.

6.107.3.5 NDDS_Transport_Address_is_ipv4()

```
RTI_INT32 NDDS_Transport_Address_is_ipv4 (
             const  NDDS_Transport_Address_t * address_in )
```

Checks if an address is an IPv4 address.

Parameters

address_in	<<**in**>> *(p. 355)* Address to be tested.

Note

May be implemented as a macro for efficiency.

Returns

1 if address is an IPv4 address, 0 otherwise.

6.107.3.6 NDDS_Transport_Address_is_multicast()

```
RTI_INT32 NDDS_Transport_Address_is_multicast (
            const  NDDS_Transport_Address_t * address_in )
```

Checks if an address is an IPv4 or IPv6 multicast address.

Parameters

address_in	<<**in**>> *(p. 355)* Address to be tested.

May be implemented as a macro for efficiency.

Returns

1 if address is a multicast address, 0 otherwise.

6.107.4 Variable Documentation

6.107.4.1 NDDS_TRANSPORT_ADDRESS_INVALID

```
const  NDDS_Transport_Address_t NDDS_TRANSPORT_ADDRESS_INVALID  [extern]
```

An invalid transport address.

6.108 UDP Transport Plugin definitions

UDP Transport Plugin definitions.

Classes

- struct **NDDS_Transport_UDP_WAN_CommPortsMappingInfo**

  Type for storing UDP WAN communication ports.

Macros

- #define **NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT** (0)

  *Default value of **NDDS_Transport_Property_t::properties_bitmap*** (p. *1788*).
- #define **NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT** (16)

  *Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max*** (p. *1789*).
- #define **NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX** RTI_OSAPI_UDP_SOCKET_↩ BUFFER_SIZE_OS_MAX

  Used to specify that os maximum be used to specify socket buffer size.
- #define **NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT** RTI_OSAPI_UDP_SOCKET↩ _BUFFER_SIZE_OS_DEFAULT

  Used to specify that os default be used to specify socket buffer size.
- #define **NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT** RTI_OSAPI_UDP_SEND↩ _SOCKET_BUFFER_SIZE_DEFAULT

  Default value of send_socket_buffer_size.
- #define **NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT** RTI_OSAPI_UDP_RECV↩ _SOCKET_BUFFER_SIZE_DEFAULT

  Default value of recv_socket_buffer_size.
- #define **NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT** (1)

  Default value of multicast_ttl.

Typedefs

- typedef RTI_UINT16 **NDDS_Transport_UDP_Port**

  UDP port.

6.108.1 Detailed Description

UDP Transport Plugin definitions.

6.108.2 Macro Definition Documentation

6.108.2.1 NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT

```
#define NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT (0)
```

Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 1788).

6.108.2.2 NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT

```
#define NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT (16)
```

Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789).

This is also the maximum value that can be used when instantiating the udp transport.

16 is sufficient for RTI Connext, but more may improve discovery and reliable performance. Porting note: find out what the maximum gather buffer count is on your OS!

6.108.2.3 NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX

```
#define NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX  RTI_OSAPI_UDP_SOCKET_BUFFER_SIZE_OS_MAX
```

Used to specify that os maximum be used to specify socket buffer size.

This property, if set, uses the maximum allowable size set in your OS for send and receive socket buffer sizes. If you update the maximum allowable size in your OS setting, then RTI Connext will automatically pick up that change. This property is not supported on all platforms because not all platforms have maximum values for the size of the send and receive buffers for sockets. If the platform does not support this property, then the property will be set to **NDDS_Transport_UDP_Plugin::NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT** (p. 379) or **NDDS_Transport_UDP_Plugin::NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT** (p. 380), depending on the property being set. See the Platform Notes to determine if this property is supported on your platform.

This value is -2.

6.108.2.4 NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT

```
#define NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT  RTI_OSAPI_UDP_SOCKET_BUFFER_SIZE_OS_↩
DEFAULT
```

Used to specify that os default be used to specify socket buffer size.

This property, if set, uses the default size set in your OS for send and receive socket buffer sizes. If you update the default size in your OS settings, then RTI Connext will automatically pick up that change.

This value is -1.

6.108.2.5 NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT  RTI_OSAPI_UDP_SEND_SOCKET_BUFFER_↩
SIZE_DEFAULT
```

Default value of send_socket_buffer_size.

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

6.108.2.6 NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT  RTI_OSAPI_UDP_RECV_SOCKET_BUFFER_↩
SIZE_DEFAULT
```

Default value of recv_socket_buffer_size.

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

6.108.2.7 NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT

```
#define NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT (1)
```

Default value of multicast_ttl.

6.108.3 Typedef Documentation

6.108.3.1 NDDS_Transport_UDP_Port

```
typedef RTI_UINT16  NDDS_Transport_UDP_Port
```

UDP port.

6.109 Shared Memory Transport

Built-in transport plug-in for inter-process communications using shared memory.

Classes

- struct **NDDS_Transport_Shmem_Property_t**

  *Subclass of **NDDS_Transport_Property_t** (p. 1786) allowing specification of parameters that are specific to the shared-memory transport.*

Macros

- #define **NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT** (-96)

  *Default value of **NDDS_Transport_Property_t::address_bit_count** (p. 1788).*
- #define **NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT** (**NDDS_TRANSPORT_↩ PROPERTY_BIT_BUFFER_ALWAYS_LOANED**)

  *Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 1788).*
- #define **NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT** (1024)

  *Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789).*
- #define **NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT** (65536)

  *Default value of **NDDS_Transport_Property_t::message_size_max** (p. 1789).*
- #define **NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT** (64)

  *Default value of **NDDS_Transport_Shmem_Property_t::received_message_count_max** (p. 1795).*
- #define **NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT**

  *Default value of **NDDS_Transport_Shmem_Property_t::receive_buffer_size** (p. 1795).*
- #define **NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX** (2)

  Major version for the transport plugin after fixing bug 14240 (RTI-28)
- #define **NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT**

  Use this to initialize stack variable.

Functions

- NDDS_Transport_Plugin ∗ **NDDS_Transport_Shmem_new** (const struct **NDDS_Transport_Shmem_↩ Property_t** ∗property_in)

  Create a new shmem process transport.

6.109.1 Detailed Description

Built-in transport plug-in for inter-process communications using shared memory.

This transport plugin uses System Shared Memory to send messages between processes on the same node. This transport is installed as a built-in transport plugin with the alias **rti::core::policy::TransportBuiltin::SHMEM_ALIAS** (p. 2536).

6.109.2 Compatibility of Sender and Receiver Transports

Opening a receiver "port" on shared memory corresponds to creating a shared memory segment using a name based on the port number. The transport plugin's properties are embedded in the shared memory segment.

When a sender tries to send to the shared memory port, it verifies that properties of the receiver's shared memory transport are compatible with those specified in its transport plugin. If not, the sender will fail to attach to the port and will output messages such as below (with numbers appropriate to the properties of the transport plugins involved).

```
NDDS_Transport_Shmem_attachShmem:failed to initialize incompatible properties

NDDS_Transport_Shmem_attachShmem:countMax 0 > -19417345 or max size -19416188 > 2147482624
```

In this scenario, the properties of the sender or receiver transport plugin instances should be adjusted, so that they are compatible.

6.109.3 Crashing and Restarting Programs

If a process using shared memory crashes (say because the user typed in ^C), resources associated with its shared memory ports may not be properly cleaned up. Later, if another RTI Connext process needs to open the same ports (say, the crashed program is restarted), it will attempt to reuse the shared memory segment left behind by the crashed process.

The reuse is allowed iff the properties of transport plugin are compatible with those embedded in the shared memory segment (i.e., of the original creator). Otherwise, the process will fail to open the ports, and will output messages such as below (with numbers appropriate to the properties of the transport plugins involved).

```
NDDS_Transport_Shmem_create_recvresource_rrEA:failed to initialize shared

    memory resource Cannot recycle existing shmem: size not compatible for key 0x1234
```

In this scenario, the shared memory segments must be cleaned up using appropriate platform specific commands. For details, please refer to the `Platform Notes`.

6.109.4 Shared Resource Keys

The transport uses the **shared memory segment keys**, given by the formula below.
`0x400000 + port`

The transport also uses signaling **shared semaphore** keys given by the formula below.
`0x800000 + port`

The transport also uses mutex **shared semaphore keys** given by the formula below.
`0xb00000 + port`

wher the `port` is a function of the `domain_id` and the `participant_id`, as described in **rti::core::policy::↩ WireProtocol::participant_id** (p. 2635)

See also

rti::core::policy::WireProtocol::participant_id (p. 2635)

Transport_Support::set_builtin_transport_property()

6.109.5 Creating and Registering Shared Memory Transport Plugin

RTI Connext can implicitly create this plugin and register with the **dds::domain::DomainParticipant** (p. 1326) if this transport is specified in **rti::core::policy::TransportBuiltin** (p. 2533).

To specify the properties of the builtin shared memory transport that is implicitly registered, you can either:

- call Transport_Support::set_builtin_transport_property or
- specify the pre-defined property names in **rti::core::policy::Property** (p. 1969) associated with the **dds↩ ::domain::DomainParticipant** (p. 1326). (see **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 383)).

Builtin transport plugin properties specified in **rti::core::policy::Property** (p. 1969) always overwrite the ones specified through Transport_Support::set_builtin_transport_property(). The default value is assumed on any unspecified property. Note that all properties should be set before the transport is implicitly created and registered by RTI Connext. See **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered.

6.109.6 Shared Memory Transport Property Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in the dds::domain::qos::DomainParticipant↩ Qos::property to to configure the builtin shared memory transport plugin.

Table 6.81 ***Property Strings for Shared Memory Transport***

Name	Descriptions
dds.transport.shmem.builtin.parent.address_bit_count	See **NDDS_Transport_Property_t::address_bit_↩ count** (p. 1788)
dds.transport.shmem.builtin.parent.properties_bitmap	See **NDDS_Transport_Property_t::properties_↩ bitmap** (p. 1788)
dds.transport.shmem.builtin.parent.gather_send_↩ buffer_count_max	See **NDDS_Transport_Property_t::gather_send_↩ buffer_count_max** (p. 1789)
dds.transport.shmem.builtin.parent.message_size_max	See **NDDS_Transport_Property_t::message_size_↩ max** (p. 1789)
dds.transport.shmem.builtin.parent.allow_interfaces	See **NDDS_Transport_Property_t::allow_↩ interfaces_list** (p. 1789) and **NDDS_Transport_↩ Property_t::allow_interfaces_list_length** (p. 1790). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.deny_interfaces	See **NDDS_Transport_Property_t::deny_interfaces↩ _list** (p. 1790) and **NDDS_Transport_Property_t↩ ::deny_interfaces_list_length** (p. 1791). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.allow_multicast_↩ interfaces	See **NDDS_Transport_Property_t::allow_multicast↩ _interfaces_list** (p. 1791) and **NDDS_Transport_↩ Property_t::allow_multicast_interfaces_list_length** (p. 1792). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.deny_multicast_↩ interfaces	See **NDDS_Transport_Property_t::deny_multicast↩ _interfaces_list** (p. 1792) and **NDDS_Transport_↩ Property_t::deny_multicast_interfaces_list_length** (p. 1792). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.max_interface_↩ count	See **NDDS_Transport_Property_t::max_interface_↩ count** (p. 1793)
dds.transport.shmem.builtin.parent.thread_name_prefix	See **NDDS_Transport_Property_t::thread_name_↩ prefix** (p. 1793)
dds.transport.shmem.builtin.received_message_↩ count_max	See ShmemTransport_Property_t::received_message↩ _count_max

Name	Descriptions
dds.transport.shmem.builtin.receive_buffer_size	See ShmemTransport_Property_t::receive_buffer_size
dds.transport.shmem.builtin.enable_udp_debugging	See ShmemTransport_Property_t::enable_udp_↩ debugging
dds.transport.shmem.builtin.udp_debugging_address	See ShmemTransport_Property_t::udp_debugging_↩ address
dds.transport.shmem.builtin.udp_debugging_port	See ShmemTransport_Property_t::udp_debugging_port

6.109.7 Macro Definition Documentation

6.109.7.1 NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT

```
#define NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT (-96)
```

Default value of **NDDS_Transport_Property_t::address_bit_count** (p. 1788).

6.109.7.2 NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT ( NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_↩
ALWAYS_LOANED)
```

Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 1788).

6.109.7.3 NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT (1024)
```

Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789).

6.109.7.4 NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT (65536)
```

Default value of **NDDS_Transport_Property_t::message_size_max** (p. 1789).

6.109.7.5 NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT (64)
```

Default value of **NDDS_Transport_Shmem_Property_t::received_message_count_max** (p. 1795).

6.109.7.6 NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT
```

Value:

```
(NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT \
 * NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT / 4)
```

Default value of **NDDS_Transport_Shmem_Property_t::receive_buffer_size** (p. 1795).

6.109.7.7 NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX

```
#define NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX (2)
```

Major version for the transport plugin after fixing bug 14240 (RTI-28)

6.109.7.8 NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT
```

Use this to initialize stack variable.

6.109.8 Function Documentation

6.109.8.1 NDDS_Transport_Shmem_new()

```
NDDS_Transport_Plugin * NDDS_Transport_Shmem_new (
            const struct  NDDS_Transport_Shmem_Property_t * property_in )
```

Create a new shmem process transport.

An application may create multiple transports, possibly for use in different domains.

Parameters

property_in	<<**in**>> *(p. 355)* Desired behavior of this transport. May be NULL for default property. The transport plugin can only support one unicast receive interface; therefore the interface selection lists are ignored.

Returns

handle to a Shmem inter-process Transport Plugin on success

NULL on failure.

6.110 UDPv4 Transport

Transport plug-in using UDP/IPv4.

Classes

- struct **NDDS_Transport_UDPv4_Property_t**

  Configurable IPv4/UDP Transport-Plugin properties.

Macros

- #define **NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT** (32)

  *Default value of **NDDS_Transport_Property_t::address_bit_count** (p. 1788).*
- #define **NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT** (128)

  *Default value of **NDDS_Transport_Property_t::address_bit_count** (p. 1788) for UDPv4 asymmetric transport.*
- #define **NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT** **NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT**

  *Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 1788).*
- #define **NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT** **NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT**

  *Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789).*
- #define **NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX** **NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX**

  OS maximum to be used to specify the socket buffer size.
- #define **NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT** **NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT**

  OS default to be used to specify the socket buffer size.
- #define **NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT** **NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT**

  *Default value for **NDDS_Transport_UDPv4_Property_t::send_socket_buffer_size** (p. 1800) and **NDDS_Transport_UDPv4_WAN_Property_t::send_socket_buffer_size** (p. 1810).*
- #define **NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT** **NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT**

*Default value for **NDDS_Transport_UDPv4_Property_t::recv_socket_buffer_size** (p. 1800) and **NDDS_Transport_↩ UDPv4_WAN_Property_t::recv_socket_buffer_size** (p. 1810).*

- #define **NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX** (65507)

  *Maximum value of **NDDS_Transport_Property_t::message_size_max** (p. 1789).*
- #define **NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT** **NDDS_TRANSPORT_UDPV4_↩ PAYLOAD_SIZE_MAX**

  *Maximum value of **NDDS_Transport_Property_t::message_size_max** (p. 1789).*
- #define **NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT** **NDDS_TRANSPORT_UDP_↩ MULTICAST_TTL_DEFAULT**

  *Default value of **NDDS_Transport_UDPv4_Property_t::multicast_ttl** (p. 1801).*
- #define **NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER**

  *Value for **NDDS_Transport_UDPv4_Property_t::send_blocking** (p. 1804) to specify non-blocking sockets.*
- #define **NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS**

  ***[default]** Value for **NDDS_Transport_UDPv4_Property_t::send_blocking** (p. 1804) to specify blocking sockets.*
- #define **NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT** NDDS_TRANSPORT_UDP_BLOCKING_↩ DEFAULT

  *Default value for **NDDS_Transport_UDPv4_Property_t::send_blocking** (p. 1804) to specify blocking sockets.*
- #define **NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT**

  *Use this to initialize a **NDDS_Transport_UDPv4_Property_t** (p. 1798) structure.*
- #define **NDDS_Transport_UDPv4_string_to_address_cEA** NDDS_Transport_UDP_string_to_address_cEA

  Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP transports.

Functions

- NDDS_Transport_Plugin * **NDDS_Transport_UDPv4_new** (const struct **NDDS_Transport_UDPv4_↩ Property_t** *property_in)

  Create an instance of a UDPv4 Transport Plugin.

6.110.1 Detailed Description

Transport plug-in using UDP/IPv4.

This transport plugin uses UDPv4 sockets to send and receive messages. It supports both unicast and multicast communications in a single instance of the plugin. By default, this plugin will use all interfaces that it finds enabled and "UP" at instantiation time to send and receive messages. This transport is installed as a built-in transport plugin with the alias **rti::core::policy::TransportBuiltin::UDPv4_ALIAS** (p. 2537).

You can configure an instance of this plugin to only use unicast or only use multicast, see **NDDS_Transport_UDPv4↩ _Property_t::unicast_enabled** (p. 1801) and **NDDS_Transport_UDPv4_Property_t::multicast_enabled** (p. 1801).

In addition, you can configure an instance of this plugin to selectively use the network interfaces of a node (and restrict a plugin from sending multicast messages on specific interfaces) by specifying the **NDDS_Transport_↩ Property_t::max_interface_count** (p. 1793) and the "white" and "black" lists in the base property's fields (**NDDS↩ _Transport_Property_t::allow_interfaces_list** (p. 1789), **NDDS_Transport_Property_t::deny_interfaces_list** (p. 1790), **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 1791), **NDDS_Transport_Property_↩ t::deny_multicast_interfaces_list** (p. 1792)).

RTI Connext can implicitly create this plugin and register with the **dds::domain::DomainParticipant** (p. 1326) if this transport is specified in **rti::core::policy::TransportBuiltin** (p. 2533).

To specify the properties of the builtin UDPv4 transport that is implicitly registered, you can either:

- call Transport_Support::set_builtin_transport_property or
- specify the predefined property names in **rti::core::policy::Property** (p. 1969) associated with the **dds::domain::DomainParticipant** (p. 1326). (see **UDPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 388)). Builtin transport plugin properties specified in **rti::core::policy::Property** (p. 1969) always overwrite the ones specified through Transport_Support::set_builtin_transport_property(). The default value is assumed on any unspecified property.

Note that all properties should be set before the transport is implicitly created and registered by RTI Connext. Any properties set after the builtin transport is registered will be ignored. See **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered.

6.110.2 UDPv4 Transport Property Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in the dds::domain::qos::DomainParticipantQos::property to configure the builtin UDPv4 transport plugin.

Table 6.83 ***Property Names for UDPv4 Transport Plugin***

Property Name	Description
dds.transport.UDPv4.builtin.parent.classid	See **NDDS_Transport_Property_t::classid** (p. 1787)
dds.transport.UDPv4.builtin.parent.address_bit_count	See **NDDS_Transport_Property_t::address_bit_count** (p. 1788)
dds.transport.UDPv4.builtin.parent.properties_bitmap	See **NDDS_Transport_Property_t::properties_bitmap** (p. 1788)
dds.transport.UDPv4.builtin.parent.gather_send_buffer_count_max	See **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789)
dds.transport.UDPv4.builtin.parent.message_size_max	See **NDDS_Transport_Property_t::message_size_max** (p. 1789)
dds.transport.UDPv4.builtin.parent.allow_interfaces	See **NDDS_Transport_Property_t::allow_interfaces_list** (p. 1789) and **NDDS_Transport_Property_t::allow_interfaces_list_length** (p. 1790). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example, 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.deny_interfaces	See **NDDS_Transport_Property_t::deny_interfaces_list** (p. 1790) and **NDDS_Transport_Property_t::deny_interfaces_list_length** (p. 1791). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.allow_multicast_interfaces	See **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 1791) and **NDDS_Transport_Property_t::allow_multicast_interfaces_list_length** (p. 1792). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0

Property Name	Description
dds.transport.UDPv4.builtin.parent.deny_multicast_↩ interfaces	See **NDDS_Transport_Property_t::deny_multicast↩ _interfaces_list** (p. 1792) and **NDDS_Transport_↩ Property_t::deny_multicast_interfaces_list_length** (p. 1792). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.max_interface_count	See **NDDS_Transport_Property_t::max_interface_↩ count** (p. 1793)
dds.transport.UDPv4.builtin.parent.thread_name_prefix	See **NDDS_Transport_Property_t::thread_name_↩ prefix** (p. 1793)
dds.transport.UDPv4.builtin.send_socket_buffer_size	See **NDDS_Transport_UDPv4_Property_t::send_↩ socket_buffer_size** (p. 1800)
dds.transport.UDPv4.builtin.recv_socket_buffer_size	See **NDDS_Transport_UDPv4_Property_t::recv_↩ socket_buffer_size** (p. 1800)
dds.transport.UDPv4.builtin.unicast_enabled	See **NDDS_Transport_UDPv4_Property_t::unicast↩ _enabled** (p. 1801)
dds.transport.UDPv4.builtin.multicast_enabled	See **NDDS_Transport_UDPv4_Property_t↩ ::multicast_enabled** (p. 1801)
dds.transport.UDPv4.builtin.multicast_ttl	See **NDDS_Transport_UDPv4_Property_t↩ ::multicast_ttl** (p. 1801)
dds.transport.UDPv4.builtin.multicast_loopback_↩ disabled	See **NDDS_Transport_UDPv4_Property_t↩ ::multicast_loopback_disabled** (p. 1802)
dds.transport.UDPv4.builtin.ignore_loopback_interface	See **NDDS_Transport_UDPv4_Property_t::ignore_↩ loopback_interface** (p. 1802)
dds.transport.UDPv4.builtin.ignore_nonrunning_↩ interfaces	See **NDDS_Transport_UDPv4_Property_t::ignore_↩ nonrunning_interfaces** (p. 1803)
dds.transport.UDPv4.builtin.ignore_nonup_interfaces	**[DEPRECATED]** See **NDDS_Transport_UDPv4_↩ Property_t::ignore_nonup_interfaces** (p. 1802)
dds.transport.UDPv4.builtin.no_zero_copy	**[DEPRECATED]** See **NDDS_Transport_UDPv4_↩ Property_t::no_zero_copy** (p. 1803)
dds.transport.UDPv4.builtin.send_blocking	See **NDDS_Transport_UDPv4_Property_t::send_↩ blocking** (p. 1804)
dds.transport.UDPv4.builtin.transport_priority_mask	See **NDDS_Transport_UDPv4_Property_t↩ ::transport_priority_mask** (p. 1804)
dds.transport.UDPv4.builtin.transport_priority_↩ mapping_low	See **NDDS_Transport_UDPv4_Property_t↩ ::transport_priority_mapping_low** (p. 1805)
dds.transport.UDPv4.builtin.transport_priority_↩ mapping_high	See **NDDS_Transport_UDPv4_Property_t↩ ::transport_priority_mapping_high** (p. 1805)
dds.transport.UDPv4.builtin.send_ping	See **NDDS_Transport_UDPv4_Property_t::send_↩ ping** (p. 1805)
dds.transport.UDPv4.builtin.force_interface_poll_↩ detection	See **NDDS_Transport_UDPv4_Property_t::force_↩ interface_poll_detection** (p. 1806)
dds.transport.UDPv4.builtin.interface_poll_period	See **NDDS_Transport_UDPv4_Property_t↩ ::interface_poll_period** (p. 1806)
dds.transport.UDPv4.builtin.reuse_multicast_receive_↩ resource	See **NDDS_Transport_UDPv4_Property_t::reuse_↩ multicast_receive_resource** (p. 1806)
dds.transport.UDPv4.builtin.protocol_overhead_max	See **NDDS_Transport_UDPv4_Property_t↩ ::protocol_overhead_max** (p. 1806)

Property Name	Description
dds.transport.UDPv4.builtin.disable_interface_tracking	See **NDDS_Transport_UDPv4_Property_t::disable_interface_tracking** (p. 1807)
dds.transport.UDPv4.builtin.public_address	See **NDDS_Transport_UDPv4_Property_t::public_address** (p. 1807)
dds.transport.UDPv4.builtin.join_multicast_group_timeout	See **NDDS_Transport_UDPv4_Property_t::join_multicast_group_timeout** (p. 1807)
dds.transport.UDPv4.builtin.port_offset	See **NDDS_Transport_UDPv4_Property_t::port_offset** (p. 1808)

See also

Transport_Support::set_builtin_transport_property()

6.110.3 Macro Definition Documentation

6.110.3.1 NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT

```
#define NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT (32)
```

Default value of **NDDS_Transport_Property_t::address_bit_count** (p. 1788).

6.110.3.2 NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT

```
#define NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT (128)
```

Default value of **NDDS_Transport_Property_t::address_bit_count** (p. 1788) for UDPv4 asymmetric transport.

6.110.3.3 NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT   NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT
```

Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 1788).

6.110.3.4 NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT NDDS_TRANSPORT_UDP_GATHER_↩
SEND_BUFFER_COUNT_MAX_DEFAULT
```

Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789).

This is also the maximum value that can be used when instantiating the UDP transport.

16 is sufficient for RTI Connext, but more may improve discovery and reliable performance. Porting note: find out what the maximum gather buffer count is on your OS!

6.110.3.5 NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX

```
#define NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_↩
OS_MAX
```

OS maximum to be used to specify the socket buffer size.

This property, if set, uses the maximum allowable size set in your OS for send and receive socket buffer sizes. If you update the maximum allowable size in your OS setting, then RTI Connext will automatically pick up that change. This property is not supported on all platforms because not all platforms have maximum values for the size of the send and receive buffers for sockets. See the `Platform Notes` to determine if this property is supported on your platform.

6.110.3.6 NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT NDDS_TRANSPORT_UDP_SOCKET_BUFFER_↩
SIZE_OS_DEFAULT
```

OS default to be used to specify the socket buffer size.

6.110.3.7 NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT NDDS_TRANSPORT_UDP_SEND_SOCKET_↩
BUFFER_SIZE_DEFAULT
```

Default value for **NDDS_Transport_UDPv4_Property_t::send_socket_buffer_size** (p. 1800) and **NDDS_Transport↩ _UDPv4_WAN_Property_t::send_socket_buffer_size** (p. 1810).

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

6.110.3.8 NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT   NDDS_TRANSPORT_UDP_RECV_SOCKET_↩
BUFFER_SIZE_DEFAULT
```

Default value for **NDDS_Transport_UDPv4_Property_t::recv_socket_buffer_size** (p. 1800) and **NDDS_Transport↩ _UDPv4_WAN_Property_t::recv_socket_buffer_size** (p. 1810).

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

6.110.3.9 NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX

```
#define NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX (65507)
```

Maximum value of **NDDS_Transport_Property_t::message_size_max** (p. 1789).

6.110.3.10 NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT   NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX
```

Maximum value of **NDDS_Transport_Property_t::message_size_max** (p. 1789).

6.110.3.11 NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT   NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT
```

Default value of **NDDS_Transport_UDPv4_Property_t::multicast_ttl** (p. 1801).

6.110.3.12 NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER

```
#define NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER
```

Value for **NDDS_Transport_UDPv4_Property_t::send_blocking** (p. 1804) to specify non-blocking sockets.

6.110.3.13 NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS

```
#define NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS
```

[default] Value for **NDDS_Transport_UDPv4_Property_t::send_blocking** (p. 1804) to specify blocking sockets.

6.110.3.14 NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT  NDDS_TRANSPORT_UDP_BLOCKING_DEFAULT
```

Default value for **NDDS_Transport_UDPv4_Property_t::send_blocking** (p. 1804) to specify blocking sockets.

6.110.3.15 NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT
```

Use this to initialize a **NDDS_Transport_UDPv4_Property_t** (p. 1798) structure.

6.110.3.16 NDDS_Transport_UDPv4_string_to_address_cEA

```
#define NDDS_Transport_UDPv4_string_to_address_cEA  NDDS_Transport_UDP_string_to_address_cEA
```

Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP transports.

Converts a host name string to a IPv4 address.

Parameters

self	NOT USED. May be NULL.
address_out	<<***out***>> *(p. 355)* The corresponding numerical value in IPv4 format.
address_in	<<***in***>> *(p. 355)* The name of the IPv4 address. It can be a dot notation name or a host name. If NULL, then the IP address of the localhost will be returned.

See also

NDDS_Transport_String_To_Address_Fcn_cEA for complete documentation.

6.110.4 Function Documentation

6.110.4.1 NDDS_Transport_UDPv4_new()

```
NDDS_Transport_Plugin * NDDS_Transport_UDPv4_new (
            const struct  NDDS_Transport_UDPv4_Property_t * property_in )
```

Create an instance of a UDPv4 Transport Plugin.

An application may create and register multiple instances of this Transport Plugin with RTI Connext. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton." For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail.

The transport plugin honors the interface/multicast "white" and "black" lists specified in the NDDS_Transport_UDP_↩ Property_t::parent:

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 1789),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 1790),
- **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 1791),
- **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 1792)

The format of a string in these lists is assumed to be in standard IPv4 dot notation, possibly containing wildcards. For example:

- 10.10.30.232
- 10.10.∗.∗
- 192.168.1.∗
- etc. Strings not in the correct format will be ignored.

Parameters

property↩ _in	<<***in***>> *(p. 355)* Desired behavior of this transport. May be NULL for default property.

Returns

A UDPv4 Transport Plugin instance on success; or NULL on failure.

6.111 Real-Time WAN Transport

Transport plug-in using UDP/IPv4 for WAN communications..

Classes

- struct **NDDS_Transport_UDPv4_WAN_Property_t**

  Configurable IPv4/UDP WAN Transport-Plugin properties.

Macros

- #define **NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT** (1400)

  *Maximum value of **NDDS_Transport_Property_t::message_size_max*** (p. *1789*).
- #define **NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT**

  *Use this to initialize a **NDDS_Transport_UDPv4_WAN_Property_t*** (p. *1809*) *structure.*

Functions

- NDDS_Transport_Plugin * **NDDS_Transport_UDPv4_WAN_new** (const struct **NDDS_Transport_UDPv4_↩ WAN_Property_t** *property_in)

  Create an instance of a UDPv4_WAN Transport Plugin.

6.111.1 Detailed Description

Transport plug-in using UDP/IPv4 for WAN communications..

RTI Real-Time WAN Transport (RWT) is a transport that enables secure, scalable, and high-performance communication over wide area networks (WANs), including public networks.

It extends RTI Connext capabilities to WAN environments. Real-Time WAN Transport uses UDPv4 as the underlying IP transport-layer protocol to better anticipate and adapt to the challenges of diverse network conditions, device mobility, and the dynamic nature of WAN system architectures.

Real-Time WAN Transport, in combination with RTI Cloud Discovery Service (CDS), provides a complete, seamless solution out of the box for WAN connectivity.

This transport is not installed as part of an RTI Connext package; it must be downloaded and installed separately.

Real-Time WAN Transport replaces the transport capabilities of the Secure WAN Transport optionally available with previous RTI Connext releases (prior to 7.0.0), and provides the following capabilities:

- NAT (Network Address Translator) traversal: Ability to communicate between DomainParticipants running in a Local Area Network (LAN) that is behind a NAT-enabled router, and DomainParticipants on the outside of the NAT across a WAN. This functionality is provided in combination with Cloud Discovery Service.

- IP mobility: Support for network transitions and changes in IP addresses in any of the DomainParticipants participating in the communication.

- Security: Secure communications between DomainParticipants using Security Plugins.

Real-Time WAN Transport does not require third-party components, such as STUN servers, or protocols like SIP to handle session establishment. Using a single API and security model, you can leverage the extensive capabilities of the RTI Connext framework and ecosystem, including tools and infrastructure services, even for real-time connectivity from edge to cloud and back in highly distributed systems that communicate across wide area networks.

This transport plugin uses UDPv4 sockets to send and receive messages. It supports unicast communications in a single instance of the plugin. By default, this plugin will use all interfaces that it finds enabled and "UP" at instantiation time to send and receive messages. This transport is installed as a built-in transport plugin with the alias **rti::core::policy::TransportBuiltin::UDPv4_WAN_ALIAS** (p. 2537).

In addition, you can configure an instance of this plugin to selectively use the network interfaces of a node by specifying the **NDDS_Transport_Property_t::max_interface_count** (p. 1793) and the "white" and "black" lists in the base property's fields (**NDDS_Transport_Property_t::allow_interfaces_list** (p. 1789), **NDDS_Transport_Property_t::deny_interfaces_list** (p. 1790)).

RTI Connext can implicitly create this plugin and register with the **dds::domain::DomainParticipant** (p. 1326) if this transport is specified in **rti::core::policy::TransportBuiltin** (p. 2533).

To specify the properties of the Real-Time WAN Transport that is implicitly registered, you can either:

- call Transport_Support::set_builtin_transport_property or
- specify the predefined property names in **rti::core::policy::Property** (p. 1969) associated with the **dds::domain::DomainParticipant** (p. 1326). (see **Real-Time WAN Transport Property** (p. 396)). Builtin transport plugin properties specified in **rti::core::policy::Property** (p. 1969) always overwrite the ones specified through Transport_Support::set_builtin_transport_property(). The default value is assumed on any unspecified property.

Note that all properties should be set before the transport is implicitly created and registered by RTI Connext. Any properties set after the builtin transport is registered will be ignored. See **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered.

For additional details on how to configure and use the Real-Time WAN Transport, see the `Core Libraries User's Manual`.

6.111.2 Real-Time WAN Transport Property

Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in the dds::domain::qos::DomainParticipantQos::property to configure the Real-Time WAN Transport plugin.

Table 6.86 ***Property Names for Real-Time WAN Transport Plugin***

Property Name	Description
dds.transport.UDPv4_WAN.builtin.parent.classid	See **NDDS_Transport_Property_t::classid** (p. 1787)
dds.transport.UDPv4_WAN.builtin.parent.address_bit_count	See **NDDS_Transport_Property_t::address_bit_count** (p. 1788)
dds.transport.UDPv4_WAN.builtin.parent.properties_bitmap	See **NDDS_Transport_Property_t::properties_bitmap** (p. 1788)

Property Name	Description
dds.transport.UDPv4_WAN.builtin.parent.gather_↩ send_buffer_count_max	See **NDDS_Transport_Property_t::gather_send_↩ buffer_count_max** (p. 1789)
dds.transport.UDPv4_WAN.builtin.parent.message_↩ size_max	See **NDDS_Transport_Property_t::message_size_↩ max** (p. 1789)
dds.transport.UDPv4_WAN.builtin.parent.allow_↩ interfaces	See **NDDS_Transport_Property_t::allow_↩ interfaces_list** (p. 1789) and **NDDS_Transport_↩ Property_t::allow_interfaces_list_length** (p. 1790). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example, 127.0.0.1,eth0
dds.transport.UDPv4_WAN.builtin.parent.deny_↩ interfaces	See **NDDS_Transport_Property_t::deny_interfaces↩ _list** (p. 1790) and **NDDS_Transport_Property_t↩ ::deny_interfaces_list_length** (p. 1791). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv4_WAN.builtin.parent.max_↩ interface_count	See **NDDS_Transport_Property_t::max_interface_↩ count** (p. 1793)
dds.transport.UDPv4_WAN.builtin.parent.thread_↩ name_prefix	See **NDDS_Transport_Property_t::thread_name_↩ prefix** (p. 1793)
dds.transport.UDPv4_WAN.builtin.send_socket_↩ buffer_size	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::send_socket_buffer_size** (p. 1810)
dds.transport.UDPv4_WAN.builtin.recv_socket_buffer↩ _size	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::recv_socket_buffer_size** (p. 1810)
dds.transport.UDPv4_WAN.builtin.ignore_loopback_↩ interface	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::ignore_loopback_interface** (p. 1811)
dds.transport.UDPv4_WAN.builtin.ignore_nonrunning↩ _interfaces	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::ignore_nonrunning_interfaces** (p. 1812)
dds.transport.UDPv4_WAN.builtin.ignore_nonup_↩ interfaces	**[DEPRECATED]** See **NDDS_Transport_UDPv4_↩ WAN_Property_t::ignore_nonup_interfaces** (p. 1811)
dds.transport.UDPv4_WAN.builtin.no_zero_copy	**[DEPRECATED]** See **NDDS_Transport_UDPv4_↩ WAN_Property_t::no_zero_copy** (p. 1812)
dds.transport.UDPv4_WAN.builtin.send_blocking	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::send_blocking** (p. 1813)
dds.transport.UDPv4_WAN.builtin.transport_priority_↩ mask	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::transport_priority_mask** (p. 1813)
dds.transport.UDPv4_WAN.builtin.transport_priority_↩ mapping_low	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::transport_priority_mapping_low** (p. 1814)
dds.transport.UDPv4_WAN.builtin.transport_priority_↩ mapping_high	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::transport_priority_mapping_high** (p. 1814)
dds.transport.UDPv4_WAN.builtin.send_ping	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::send_ping** (p. 1814)
dds.transport.UDPv4_WAN.builtin.force_interface_↩ poll_detection	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::force_interface_poll_detection** (p. 1815)
dds.transport.UDPv4_WAN.builtin.interface_poll_period	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::interface_poll_period** (p. 1815)
dds.transport.UDPv4_WAN.builtin.protocol_overhead↩ _max	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::protocol_overhead_max** (p. 1815)

Property Name	Description
dds.transport.UDPv4_WAN.builtin.disable_interface_↩ tracking	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::disable_interface_tracking** (p. 1815)
dds.transport.UDPv4_WAN.builtin.public_address	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::public_address** (p. 1816)
dds.transport.UDPv4_WAN.builtin.comm_ports	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::comm_ports_list** (p. 1816)
dds.transport.UDPv4_WAN.builtin.port_offset	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::port_offset** (p. 1818)
dds.transport.UDPv4_WAN.builtin.binding_ping_period	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::binding_ping_period** (p. 1818)

See also

Transport_Support::set_builtin_transport_property()

6.111.3 Macro Definition Documentation

6.111.3.1 NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT (1400)
```

Maximum value of **NDDS_Transport_Property_t::message_size_max** (p. 1789).

6.111.3.2 NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT
```

Use this to initialize a **NDDS_Transport_UDPv4_WAN_Property_t** (p. 1809) structure.

6.111.4 Function Documentation

6.111.4.1 NDDS_Transport_UDPv4_WAN_new()

```
NDDS_Transport_Plugin * NDDS_Transport_UDPv4_WAN_new (
            const struct  NDDS_Transport_UDPv4_WAN_Property_t * property_in )
```

Create an instance of a UDPv4_WAN Transport Plugin.

An application may create and register multiple instances of this Transport Plugin with RTI Connext. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton." For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail.

The transport plugin honors the interface "white" and "black" lists specified in the NDDS_Transport_UDP_Property_t↩ ::parent:

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 1789),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 1790),

The format of a string in these lists is assumed to be in standard IPv4 dot notation, possibly containing wildcards. For example:

- 10.10.30.232
- 10.10.∗.∗
- 192.168.1.∗
- etc. Strings not in the correct format will be ignored.

Parameters

property↩ _in	<<***in***>> *(*p. *355)* Desired behavior of this transport. May be NULL for default property.

Returns

A UDPv4_WAN Transport Plugin instance on success; or NULL on failure.

6.112 UDPv6 Transport

Transport plug-in using UDP/IPv6.

Classes

- struct **NDDS_Transport_UDPv6_Property_t**

  Configurable IPv6/UDP Transport-Plugin properties.

Macros

- #define **NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT** (128)

  *Default value of **NDDS_Transport_Property_t::address_bit_count*** (p. *1788*).
- #define **NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT** **NDDS_TRANSPORT_UDP_↩ PROPERTIES_BITMAP_DEFAULT**

  *Default value of **NDDS_Transport_Property_t::properties_bitmap*** (p. *1788*).
- #define **NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT** **NDDS_↩ TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT**

  *Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max*** (p. *1789*).
- #define **NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX** **NDDS_TRANSPORT_UDP_↩ SOCKET_BUFFER_SIZE_OS_MAX**

  OS maximum to be used to specify the socket buffer size.
- #define **NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT** **NDDS_TRANSPORT_↩ UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT**

  OS default to be used to specify the socket buffer size.
- #define **NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT** **NDDS_TRANSPORT_↩ UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT**

  *Default value for **NDDS_Transport_UDPv6_Property_t::send_socket_buffer_size*** (p. *1820*).
- #define **NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT** **NDDS_TRANSPORT_↩ UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT**

  *Default value for **NDDS_Transport_UDPv6_Property_t::recv_socket_buffer_size*** (p. *1821*).
- #define **NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX** (65487)

  *Maximum value of **NDDS_Transport_Property_t::message_size_max*** (p. *1789*).
- #define **NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT** **NDDS_TRANSPORT_UDPV6_↩ PAYLOAD_SIZE_MAX**

  *Default value of **NDDS_Transport_Property_t::message_size_max*** (p. *1789*).
- #define **NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT** **NDDS_TRANSPORT_UDP_↩ MULTICAST_TTL_DEFAULT**

  *Default value of **NDDS_Transport_UDPv6_Property_t::multicast_ttl*** (p. *1822*).
- #define **NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER**

  *Value for **NDDS_Transport_UDPv6_Property_t::send_blocking*** (p. *1824*) *to specify non-blocking sockets.*
- #define **NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS**

  [default]** Value for **NDDS_Transport_UDPv6_Property_t::send_blocking (p. *1824*) *to specify blocking sockets.*
- #define **NDDS_Transport_UDPv6_string_to_address_cEA** NDDS_Transport_UDP_string_to_address_cEA

  Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP transports.

Functions

- NDDS_Transport_Plugin * **NDDS_Transport_UDPv6_new** (const struct **NDDS_Transport_UDPv6_↩ Property_t** *property_in)

  Create an instance of a UDPv6 Transport Plugin.

6.112.1 Detailed Description

Transport plug-in using UDP/IPv6.

This transport plugin uses UDPv6 sockets to send and receive messages. It supports both unicast and multicast communications in a single instance of the plugin. By default, this plugin will use all interfaces that it finds enabled and "UP" at instantiation time to send and receive messages. This transport is installed as a built-in transport plugin with the alias **rti::core::policy::TransportBuiltin::UDPv6_ALIAS** (p. 2537).

You can configure an instance of this plugin to only use unicast or only use multicast, see **NDDS_Transport_UDPv6_Property_t::unicast_enabled** (p. 1821) and **NDDS_Transport_UDPv6_Property_t::multicast_enabled** (p. 1822).

In addition, you can configure an instance of this plugin to selectively use the network interfaces of a node (and restrict a plugin from sending multicast messages on specific interfaces) by specifying the **NDDS_Transport_Property_t::max_interface_count** (p. 1793) and the "white" and "black" lists in the base property's fields (**NDDS_Transport_Property_t::allow_interfaces_list** (p. 1789), **NDDS_Transport_Property_t::deny_interfaces_list** (p. 1790), **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 1791), **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 1792)).

RTI Connext can implicitly create this plugin and register it with the **dds::domain::DomainParticipant** (p. 1326) if this transport is specified in the **rti::core::policy::TransportBuiltin** (p. 2533).

To specify the properties of the builtin UDPv6 transport that is implicitly registered, you can either:

- call Transport_Support::set_builtin_transport_property or
- specify the predefined property names in **rti::core::policy::Property** (p. 1969) associated with the **dds::domain::DomainParticipant** (p. 1326). (see **UDPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 401)). Builtin transport plugin properties specified in **rti::core::policy::Property** (p. 1969) always overwrite the ones specified through Transport_Support::set_builtin_transport_property(). The default value is assumed on any unspecified property.

Note that all properties should be set before the transport is implicitly created and registered by RTI Connext. Any properties that are set after the builtin transport is registered will be ignored. See **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered.

6.112.2 UDPv6 Transport Property Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in **rti::core::policy::Property** (p. 1969) of a **dds::domain::DomainParticipant** (p. 1326) to configure the builtin UDPv6 transport plugin.

Table 6.88 ***Property Names for UDPv6 Transport Plugin***

Property Name	Description
dds.transport.UDPv6.builtin.parent.address_bit_count	See **NDDS_Transport_Property_t::address_bit_count** (p. 1788)
dds.transport.UDPv6.builtin.parent.properties_bitmap	See **NDDS_Transport_Property_t::properties_bitmap** (p. 1788)
dds.transport.UDPv6.builtin.parent.gather_send_buffer_count_max	See **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789)

Property Name	Description
dds.transport.UDPv6.builtin.parent.message_size_max	See **NDDS_Transport_Property_t::message_size_max** (p. 1789)
dds.transport.UDPv6.builtin.parent.allow_interfaces	See **NDDS_Transport_Property_t::allow_interfaces_list** (p. 1789) and **NDDS_Transport_Property_t::allow_interfaces_list_length** (p. 1790). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv6.builtin.parent.deny_interfaces	See **NDDS_Transport_Property_t::deny_interfaces_list** (p. 1790) and **NDDS_Transport_Property_t::deny_interfaces_list_length** (p. 1791). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv6.builtin.parent.allow_multicast_interfaces	See **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 1791) and **NDDS_Transport_Property_t::allow_multicast_interfaces_list_length** (p. 1792). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv6.builtin.parent.deny_multicast_interfaces	See **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 1792) and **NDDS_Transport_Property_t::deny_multicast_interfaces_list_length** (p. 1792). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv6.builtin.parent.max_interface_count	See **NDDS_Transport_Property_t::max_interface_count** (p. 1793)
dds.transport.UDPv6.builtin.parent.thread_name_prefix	See **NDDS_Transport_Property_t::thread_name_prefix** (p. 1793)
dds.transport.UDPv6.builtin.send_socket_buffer_size	See **NDDS_Transport_UDPv6_Property_t::send_socket_buffer_size** (p. 1820)
dds.transport.UDPv6.builtin.recv_socket_buffer_size	See **NDDS_Transport_UDPv6_Property_t::recv_socket_buffer_size** (p. 1821)
dds.transport.UDPv6.builtin.unicast_enabled	See **NDDS_Transport_UDPv6_Property_t::unicast_enabled** (p. 1821)
dds.transport.UDPv6.builtin.multicast_enabled	See **NDDS_Transport_UDPv6_Property_t::multicast_enabled** (p. 1822)
dds.transport.UDPv6.builtin.multicast_ttl	See **NDDS_Transport_UDPv6_Property_t::multicast_ttl** (p. 1822)
dds.transport.UDPv6.builtin.multicast_loopback_disabled	See **NDDS_Transport_UDPv6_Property_t::multicast_loopback_disabled** (p. 1822)
dds.transport.UDPv6.builtin.ignore_loopback_interface	See **NDDS_Transport_UDPv6_Property_t::ignore_loopback_interface** (p. 1822)
dds.transport.UDPv6.builtin.ignore_nonrunning_interfaces	See **NDDS_Transport_UDPv6_Property_t::ignore_nonrunning_interfaces** (p. 1823)
dds.transport.UDPv6.builtin.no_zero_copy	**[DEPRECATED]** See **NDDS_Transport_UDPv6_Property_t::no_zero_copy** (p. 1823)

Property Name	Description
dds.transport.UDPv6.builtin.send_blocking	See **NDDS_Transport_UDPv6_Property_t::send_blocking** (p. 1824)
dds.transport.UDPv6.builtin.enable_v4mapped	See **NDDS_Transport_UDPv6_Property_t::enable_v4mapped** (p. 1824)
dds.transport.UDPv6.builtin.transport_priority_mask	See **NDDS_Transport_UDPv6_Property_t::transport_priority_mask** (p. 1824)
dds.transport.UDPv6.builtin.transport_priority_mapping_low	See **NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_low** (p. 1825)
dds.transport.UDPv6.builtin.transport_priority_mapping_high	See **NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_high** (p. 1825)
dds.transport.UDPv6.builtin.send_ping	See **NDDS_Transport_UDPv6_Property_t::send_ping** (p. 1825) interface_poll_detection_kind
dds.transport.UDPv6.builtin.force_interface_poll_detection	See **NDDS_Transport_UDPv6_Property_t::force_interface_poll_detection** (p. 1826)
dds.transport.UDPv6.builtin.interface_poll_period	See **NDDS_Transport_UDPv6_Property_t::interface_poll_period** (p. 1826)
dds.transport.UDPv6.builtin.reuse_multicast_receive_resource	See **NDDS_Transport_UDPv6_Property_t::reuse_multicast_receive_resource** (p. 1826)
dds.transport.UDPv6.builtin.protocol_overhead_max	See **NDDS_Transport_UDPv6_Property_t::protocol_overhead_max** (p. 1826)
dds.transport.UDPv6.builtin.disable_interface_tracking	See **NDDS_Transport_UDPv6_Property_t::disable_interface_tracking** (p. 1827)
dds.transport.UDPv6.builtin.public_address	See **NDDS_Transport_UDPv6_Property_t::public_address** (p. 1827)
dds.transport.UDPv6.builtin.join_multicast_group_timeout	See **NDDS_Transport_UDPv6_Property_t::join_multicast_group_timeout** (p. 1827)
dds.transport.UDPv6.builtin.port_offset	See **NDDS_Transport_UDPv6_Property_t::port_offset** (p. 1828)

See also

Transport_Support::set_builtin_transport_property()

6.112.3 Macro Definition Documentation

6.112.3.1 NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT

```
#define NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT (128)
```

Default value of **NDDS_Transport_Property_t::address_bit_count** (p. 1788).

6.112.3.2 NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT   NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_↩
DEFAULT
```

Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 1788).

6.112.3.3 NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT   NDDS_TRANSPORT_UDP_GATHER_↩
SEND_BUFFER_COUNT_MAX_DEFAULT
```

Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789).

This is also the maximum value that can be used when instantiating the UDP transport.

16 is sufficient for NDDS, but more may improve discovery and reliable performance. Porting note: find out what the maximum gather buffer count is on your OS!

6.112.3.4 NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX

```
#define NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX   NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_↩
OS_MAX
```

OS maximum to be used to specify the socket buffer size.

This property, if set, uses the maximum allowable size set in your OS for send and receive socket buffer sizes. If you update the maximum allowable size in your OS setting, then RTI Connext will automatically pick up that change. This property is not supported on all platforms because not all platforms have maximum values for the size of the send and receive buffers for sockets. See the `Platform Notes` to determine if this property is supported on your platform.

6.112.3.5 NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT   NDDS_TRANSPORT_UDP_SOCKET_BUFFER_↩
SIZE_OS_DEFAULT
```

OS default to be used to specify the socket buffer size.

6.112.3.6 NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT   NDDS_TRANSPORT_UDP_SEND_SOCKET_↩
BUFFER_SIZE_DEFAULT
```

Default value for **NDDS_Transport_UDPv6_Property_t::send_socket_buffer_size** (p. 1820).

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

6.112.3.7 NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT   NDDS_TRANSPORT_UDP_RECV_SOCKET_↩
BUFFER_SIZE_DEFAULT
```

Default value for **NDDS_Transport_UDPv6_Property_t::recv_socket_buffer_size** (p. 1821).

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

6.112.3.8 NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX

```
#define NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX (65487)
```

Maximum value of **NDDS_Transport_Property_t::message_size_max** (p. 1789).

6.112.3.9 NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT   NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX
```

Default value of **NDDS_Transport_Property_t::message_size_max** (p. 1789).

6.112.3.10 NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT   NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT
```

Default value of **NDDS_Transport_UDPv6_Property_t::multicast_ttl** (p. 1822).

6.112.3.11 NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER

```
#define NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER
```

Value for **NDDS_Transport_UDPv6_Property_t::send_blocking** (p. 1824) to specify non-blocking sockets.

6.112.3.12 NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS

```
#define NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS
```

[default] Value for **NDDS_Transport_UDPv6_Property_t::send_blocking** (p. 1824) to specify blocking sockets.

6.112.3.13 NDDS_Transport_UDPv6_string_to_address_cEA

```
#define NDDS_Transport_UDPv6_string_to_address_cEA  NDDS_Transport_UDP_string_to_address_cEA
```

Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP transports.

Converts a host name string to a IPv6 address.

Parameters

self	NOT USED. May be NULL.
address_out	<<***out***>> *(*p. *355)* The corresponding numerical value in IPv6 format.
address_in	<<***in***>> *(*p. *355)* The name of the IPv6 address. It can be a dot notation name or a host name. If NULL, then the IP address of the localhost will be returned.

See also

NDDS_Transport_String_To_Address_Fcn_cEA for complete documentation.

6.112.4 Function Documentation

6.112.4.1 NDDS_Transport_UDPv6_new()

```
NDDS_Transport_Plugin * NDDS_Transport_UDPv6_new (
            const struct  NDDS_Transport_UDPv6_Property_t * property_in )
```

Create an instance of a UDPv6 Transport Plugin.

An application may create and register multiple instances of this Transport Plugin with RTI Connext. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton." For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail.

The transport plugin honors the interface/multicast "white" and "black" lists specified in the **NDDS_Transport_UDPv6_Property_t::parent** (p. 1820):

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 1789),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 1790),
- **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 1791),
- **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 1792)

The format of a string in these lists is assumed to be in standard IPv6 dot notation, possibly containing wildcards. For example:

- 10.10.30.232
- 10.10.*.*
- 192.168.1.*
- etc. Strings not in the correct format will be ignored.

Parameters

property_in	<<***in***>> *(p. 355)* Desired behavior of this transport. May be NULL for default property.

Returns

A UDPv6 Transport Plugin instance on success; or NULL on failure.

6.113 Client-side API

Part of the RPC API that relates to the client.

Classes

- class **dds::rpc::ClientEndpoint**< **Request, Reply** >

  <<***reference-type***>> *(p. 351) Manages the DDS entities required to make remote function calls.*
- class **rpc_example::RobotControlClient**

  <<***reference-type***>> *(p. 351) Allows client applications to make remote function calls*
- class **dds::rpc::ClientParams**

  <<***value-type***>> *(p. 350) The parameters used to configure a* ***ClientEndpoint*** *(p. 925)*

Typedefs

- using **rpc_example::RobotControlClientEndpoint** = **dds::rpc::ClientEndpoint**< RobotControl_Call, RobotControl_Return >

  An instantiation of ***dds::rpc::ClientEndpoint*** *(p. 925) for the DDS types that allow making function calls to the* ***Robot Control*** *(p. 2213) service interface.*

6.113.1 Detailed Description

Part of the RPC API that relates to the client.

6.113.2 Typedef Documentation

6.113.2.1 RobotControlClientEndpoint

```
using rpc_example::RobotControlClientEndpoint = typedef dds::rpc::ClientEndpoint<RobotControl_Call, RobotControl_Return>
```

An instantiation of **dds::rpc::ClientEndpoint** (p. 925) for the DDS types that allow making function calls to the **Robot Control** (p. 2213) service interface.

6.114 Server-side API

Part of the RPC API that relates to the **Server** (p. 2346) and **ServiceEndpoint** (p. 2354).

Classes

- class **dds::rpc::ServiceEndpoint**< **Dispatcher** >

  Manages the DDS entities required to receive function calls and send the return values.
- class **dds::rpc::Server**

  <<***reference-type***>> *(p. 351) Provides the execution environment for one or more* ***ServiceEndpoint*** *(p. 2354).*
- class **dds::rpc::ServerParams**

  <<***value-type***>> *(p. 350) The parameters used to configure a* ***Server*** *(p. 2346)*

Typedefs

- using **rpc_example::RobotControlService** = **dds::rpc::ServiceEndpoint**< RobotControlDispatcher >

  Executes a ***RobotControl*** *(p. 2213) interface implementation.*
- using **dds::rpc::ServiceParams** = **rti::request::ReplierParams**

  The parameters used to configure a ***ServiceEndpoint*** *(p. 2354).*

6.114.1 Detailed Description

Part of the RPC API that relates to the **Server** (p. 2346) and **ServiceEndpoint** (p. 2354).

6.114.2 Typedef Documentation

6.114.2.1 RobotControlService

```
using rpc_example::RobotControlService = typedef dds::rpc::ServiceEndpoint<RobotControlDispatcher>
```

Executes a **RobotControl** (p. 2213) interface implementation.

6.114.2.2 ServiceParams

```
typedef dds::rpc::detail::ServiceParams dds::rpc::ServiceParams
```

The parameters used to configure a **ServiceEndpoint** (p. 2354).

6.115 FlatData Builders

A Builder allows creating and initializing variable-size data.

Classes

- class **rti::flat::AggregationBuilder**
  Base class of struct and union builders.
- class **rti::flat::UnionBuilder< Discriminator >**
  Base class of builders for user-defined mutable unions.
- class **MyFlatMutableBuilder**
  Represents the Builder for an arbitrary user-defined mutable type.
- class **MyFlatUnionBuilder**
  Represents the Builder for an arbitrary user-defined mutable IDL union.
- class **rti::flat::AbstractBuilder**
  *Base class of all **Builders** (p. 409).*
- class **rti::flat::AbstractListBuilder< DHeaderGen >**
  Base class of all array and sequence builders.
- class **rti::flat::MutableArrayBuilder< ElementBuilder, N >**
  Builds an array member of variable-size elements.
- class **rti::flat::AbstractSequenceBuilder< DHeaderGen >**
  Base class of Builders for sequence members.
- class **rti::flat::MutableSequenceBuilder< ElementBuilder >**
  Builds a sequence member of variable-size elements.
- class **rti::flat::FinalSequenceBuilder< ElementOffset >**
  Builds a sequence member of fixed-size elements.
- class **rti::flat::PrimitiveSequenceBuilder< T >**
  Builds a sequence of primitive members.
- class **rti::flat::StringBuilder**
  Builds a string.

Functions

- template<typename TopicType >
  rti::flat::flat_type_traits< TopicType >::builder **rti::flat::build_data** (**dds::pub::DataWriter**< TopicType > &writer)
  Begins building a new sample.
- template<typename TopicType >
  void **rti::flat::discard_builder** (**dds::pub::DataWriter**< TopicType > &writer, typename **rti::flat::flat_type_↩ traits**< TopicType >::builder &builder)
  Discards a sample builder.

6.115.1 Detailed Description

A Builder allows creating and initializing variable-size data.

Builders allow creating variable-size **FlatData samples** (p. 412), as described in **Publishing FlatData** (p. 420).

There are the following Builder types:

Category	Builder type
User types	For example: • **MyFlatMutableBuilder** (p. 1763), a mutable struct • **MyFlatUnionBuilder** (p. 1777), a union • Final structs are fixed-size and do not have a Builder (see for example **MyFlatMutable↩ Builder::add_my_final()** (p. 1768)).
Arrays	• **rti::flat::MutableArrayBuilder** (p. 1753) • Arrays of final elements are fixed-size and do not have a Builder (see for example **MyFlat↩ MutableBuilder::add_my_final_array()** (p. 1768)).
Sequences	• **rti::flat::MutableSequenceBuilder** (p. 1757) for sequences of mutable elements • **rti::flat::FinalSequenceBuilder** (p. 1569) for sequences of final elements • **rti::flat::PrimitiveSequenceBuilder** (p. 1955) for sequences of primitive elements • **rti::flat::StringBuilder** (p. 2392)
Primitive types	• Primitive members are added using one of the member functions of the Builder of the type that contains them. See for example **MyFlatMutableBuilder::add_my_primitive()** (p. 1767)<

Builder for user types, such as **MyFlatMutableBuilder** (p. 1763), can behave as **sample builders**, when they build a sample, or **member builders** when they build a member for another Builder (such as the Builder returned by **MyFlat↩ MutableBuilder::build_my_mutable()** (p. 1769)). All the other Builder types (sequences, arrays) are always member builders (such as the Builder returned by **MyFlatMutableBuilder::build_my_final_seq()** (p. 1768)).

Builders are **move-only** types. They cannot be copied, only moved. When you assign a Builder as the return value of a function, this happens automatically.

```
// the return value of 'build_data' is moved into 'builder'
MyFlatMutableBuilder builder = rti::flat::build_data(writer);
```

To explicitly move a Builder variable use `std::move()` in C++11 platforms or call the **`move()`** (p. 1669) member function in pre-C++11 platforms.

6.115.2 Builder Error Management

In the Modern C++ API errors are notified with exceptions. Builder operations may throw the following exceptions

- **dds::core::PreconditionNotMetError** (p. 1941), for any precondition failure.
- **dds::core::OutOfResourcesError** (p. 1898), when the Builder's underlying buffer runs out of space (this should not happen when the Builder has been created with **rti::flat::build_data()** (p. 411)).

See also **Offset Error Management** (p. 416).

See also

MyFlatMutableBuilder (p. 1763) for more information specific to builders for user types.

6.115.3 Function Documentation

6.115.3.1 build_data()

```
template<typename TopicType >
rti::flat::flat_type_traits< TopicType >::builder rti::flat::build_data (
              dds::pub::DataWriter< TopicType > & writer )
```

Begins building a new sample.

Template Parameters

TopicType	A FlatData mutable type that corresponds to the type of the DataWriter argument.

Parameters

writer	The writer that will be used to write this sample.

Returns

The Builder to build the sample. For example if TopicType is **MyFlatMutable** (p. 413), this function returns a **MyFlatMutableBuilder** (p. 1763).

Once you have completed the sample, call **finish_sample()** (p. 1766) to obtain a **MyFlatMutable** (p. 413) sample that can be written with `writer`.

If the building of this sample needs to be aborted before calling finish_sample(), use **rti::flat::discard_builder()** (p. 411). If, after obtaining a sample with finish_sample(), this sample is not written, then discard it with **dds::pub::DataWriter::discard_loan()** (p. 1185).

See also

dds::pub::DataWriter::get_loan (p. 1183), the function that **build_data()** (p. 411) uses to obtain the memory required to build the sample.

Publishing FlatData (p. 420)

References **dds::pub::DataWriter< T >::get_loan()**.

6.115.3.2 discard_builder()

```
template<typename TopicType >
void rti::flat::discard_builder (
              dds::pub::DataWriter< TopicType > & writer,
             typename  rti::flat::flat_type_traits< TopicType >::builder & builder )
```

Discards a sample builder.

Template Parameters

TopicType	An IDL-defined FlatData type that corresponds to the type of the DataWriter argument.

This function invalidates and discards a builder before it is finished() and before the sample it would have created was written.

Parameters

writer	The writer that was used to create the builder
builder	The builder, created with `rti::flat::build_data(writer);`

References **dds::pub::DataWriter< T >::discard_loan()**.

6.116 FlatData Samples

A Sample represents an instance of the IDL topic-type and contains the data in serialized format.

Classes

- class **rti::flat::Sample< OffsetType >**
  The generic definition of FlatData topic-types.
- struct **rti::flat::flat_type_traits< T >**
  *Given a **Sample** (p. 2274), an Offset or a Builder, it allows obtaining the other types.*

Typedefs

- typedef **rti::flat::Sample< MyFlatFinalOffset > MyFlatFinal**
  Represents an arbitrary user-defined FlatData final IDL struct.
- typedef **rti::flat::Sample< MyFlatMutableOffset > MyFlatMutable**
  Represents an arbitrary user-defined flat mutable IDL struct.
- typedef **rti::flat::Sample< MyFlatUnionOffset > MyFlatUnion**
  Represents an arbitrary user-defined flat mutable IDL union.

6.116.1 Detailed Description

A Sample represents an instance of the IDL topic-type and contains the data in serialized format.

All FlatData topic-types are instantiations of **rti::flat::Sample** (p. 2274). This documentation uses the following three example types to illustrate the different ways FlatData IDL types map to C++:

- **MyFlatFinal** (p. 413), the type generated for a final IDL struct
- **MyFlatMutable** (p. 413), the type generated for a mutable IDL struct
- **MyFlatUnion** (p. 413), the type generated for an IDL union

6.116.2 Typedef Documentation

6.116.2.1 MyFlatFinal

```
typedef rti::flat::Sample< MyFlatFinalOffset> MyFlatFinal
```

Represents an arbitrary user-defined FlatData final IDL struct.

This documentation uses the following example IDL definition of MyFlatFinal:

```
@language_binding(FLAT_DATA)
@final
struct MyFlatFinal {
    long my_primitive;
    FlatFinalBar my_complex; // Another arbitrary final FlatData type
    long my_primitive_array[10];
    FlatFinalBar my_complex_array[10];
};
```

For this type, **rtiddsgen** generates:

- MyFlatFinal (this instantiation of Sample),
- **MyFlatFinalOffset** (p. 1759).

Note that, as a final FlatData type, MyFlatFinal can only contain fixed-size types, such as primitives, other final FlatData structs, and arrays of fixed-size types.

Samples of MyFlatFinal are created with **dds::pub::DataWriter::get_loan** (p. 1183). After creating a final Sample, modify its values using the **MyFlatFinalOffset** (p. 1759) returned by **root()** (p. 2276).

6.116.2.2 MyFlatMutable

```
typedef rti::flat::Sample< MyFlatMutableOffset> MyFlatMutable
```

Represents an arbitrary user-defined flat mutable IDL struct.

This documentation uses the following example IDL definition of MyFlatMutable:

```
@language_binding(FLAT_DATA)
@mutable
struct MyFlatMutable {
    long my_primitive;
    @optional long my_optional_primitive;
    long my_primitive_array[10];
    sequence<long, 10> my_primitive_seq;
    MyFlatFinal my_final;
    MyFlatFinal my_final_array[10];
    sequence<MyFlatFinal, 10> my_final_seq;
    FlatMutableBar my_mutable;
    FlatMutableBar my_mutable_array[10];
    sequence<FlatMutableBar, 10> my_mutable_seq;
    string<255> my_string;
    sequence<string<255>, 10> my_string_seq;
};
```

For this type, **rtiddsgen** generates:

- MyFlatMutable (this instantiation of Sample),
- **MyFlatMutableOffset** (p. 1770),
- **MyFlatMutableBuilder** (p. 1763).

As a mutable FlatData type, MyFlatMutable is not restricted to fixed-size members. Samples of MyFlatMutable are created with **rti::flat::build_data()** (p. 411), which returns a **MyFlatMutableBuilder** (p. 1763). Once built, a mutable Sample can't change in size, but the value of members that already exist can be changed using the **MyFlatMutable↩ Offset** (p. 1770) returned by **root()** (p. 2276).

6.116.2.3 MyFlatUnion

```
typedef rti::flat::Sample< MyFlatUnionOffset> MyFlatUnion
```

Represents an arbitrary user-defined flat mutable IDL union.

This documentation uses the following example IDL definition of MyFlatUnion:

```
@language_binding(FLAT_DATA)
@mutable
union MyFlatUnion switch (long) {
    case 0:
        long my_primitive;
    case 1:
    case 2:
        MyFlatMutable my_mutable;
    case 3:
        MyFlatFinal my_final;
};
```

Note

FlatData unions can only be **mutable** since unions are, by definition, variable-size types.

For this type, **rtiddsgen** generates:

- MyFlatUnion (this instantiation of Sample),
- **MyFlatUnionOffset** (p. 1780),
- **MyFlatUnionBuilder** (p. 1777).

6.117 FlatData Offsets

Offsets provide access to the members of a FlatData Sample.

Classes

- class **MyFlatFinalOffset**

  Represents the Offset to an arbitrary user-defined FlatData final IDL struct.
- class **MyFlatMutableOffset**

  Represents the Offset to an arbitrary user-defined FlatData mutable IDL struct.
- class **MyFlatUnionOffset**

  Represents the Offset to an arbitrary user-defined FlatData mutable IDL union.
- class **rti::flat::OffsetBase**

  Base class of all Offset types.
- class **rti::flat::FinalOffset**< **T** >

  The base class of all Offsets to a final struct type.
- class **rti::flat::MutableOffset**

  The base class of all Offsets to a final struct type.
- struct **rti::flat::PrimitiveConstOffset**< **T** >

  A const Offset to an optional primitive member.

- struct **rti::flat::PrimitiveOffset< T >**
  An Offset to an optional primitive member.
- class **rti::flat::SequenceIterator< E, OffsetKind >**
  Iterator for collections of Offsets.
- class **rti::flat::AbstractPrimitiveList< T >**
  Base class for Offsets to sequences and arrays of primitive types.
- class **rti::flat::PrimitiveSequenceOffset< T >**
  Offset to a sequence of primitive elements.
- class **rti::flat::PrimitiveArrayOffset< T, N >**
  Offset to an array of primitive elements.
- class **rti::flat::StringOffset**
  Offset to a string.
- class **rti::flat::AbstractAlignedList< ElementOffset >**
  Base class of Offsets to sequences and arrays of non-primitive members.
- class **rti::flat::SequenceOffset< ElementOffset >**
  Offset to a sequence of non-primitive elements.
- class **rti::flat::MutableArrayOffset< ElementOffset, N >**
  Offset to an array of variable-size elements.
- class **rti::flat::FinalArrayOffset< ElementOffset, N >**
  Offset to an array of final elements.
- class **rti::flat::FinalAlignedArrayOffset< ElementOffset, N >**
  Offset to an array of final elements.

Functions

- template<typename OffsetType >
  flat_type_traits< OffsetType >::plain_type * **rti::flat::plain_cast** (OffsetType &offset)
  Casts into an equivalent plain C++ type.
- template<typename OffsetType >
  const **flat_type_traits**< OffsetType >::plain_type * **rti::flat::plain_cast** (const OffsetType &offset)
  Const version of plain_cast.

6.117.1 Detailed Description

Offsets provide access to the members of a FlatData Sample.

An Offset represents a position within a **FlatData Sample** (p. 412) that allows accessing a member of that Sample, or the Sample's **root** (p. 2276).

Offsets can be described as **iterators**. They represent a position in a buffer, not the value itself. As such, they're lightweight objects that can be copied to point to the same data.

There are the following Offset types to access the different IDL types:

Category	Offset type
User types	For example: • **MyFlatFinalOffset** (p. 1759), an Offset to a final struct • **MyFlatMutableOffset** (p. 1770), an Offset to a mutable struct • **MyFlatUnionOffset** (p. 1780), an Offset to a union
Arrays	• **rti::flat::PrimitiveArrayOffset** (p. 1951) • **rti::flat::FinalArrayOffset** (p. 1567) (array of final elements in a final type) • **rti::flat::FinalAlignedArrayOffset** (p. 1565) (array of final elements in a mutable type) • **rti::flat::MutableArrayOffset** (p. 1755)
Sequences	• **rti::flat::PrimitiveSequenceOffset** (p. 1958) • **rti::flat::SequenceOffset** (p. 2343)
Primitive types	• The type itself, such as `double` • **rti::flat::PrimitiveOffset** (p. 1953) (when the member is optional)

Offsets for user types are generated by **rtiddsgen** and provide methods to access the type's members by their names. Offsets for arrays and sequences provide methods to access each element.

Some offsets allow accessing the data through a pointer to the equivalent plain C++ type, which generally provides better performance. See **rti::flat::plain_cast()** (p. 417).

6.117.2 Offset Error Management

Functions that return an Offset may return a "null" Offset (one such that **is_null()** (p. 1876) returns `true`).

An Offset may be null if a member doesn't exist in the **Sample** (p. 412). For example, if the member 'my_final' in **MyFlatMutable** (p. 413) doesn't exist (because it wasn't added while building the Sample, or because it wasn't received in the subscribing application), **MyFlatMutableOffset::my_final()** (p. 1775) returns an null Offset. Note that a **member in a final type** (for example, **MyFlatFinalOffset::my_complex()** (p. 1762)) always exists, except in the case of an error.

Other than that, any error condition will cause one of the following exceptions, not a null Offset:

- **dds::core::NullReferenceError** (p. 1870), when accessing a null Offset
- **dds::core::PreconditionNotMetError** (p. 1941), for any precondition failure.
- **dds::core::OutOfResourcesError** (p. 1898), when a **Builder** (p. 409) runs out of space while adding a member.

6.117.3 Function Documentation

6.117.3.1 plain_cast() [1/2]

```
template<typename OffsetType >
flat_type_traits< OffsetType >::plain_type * rti::flat::plain_cast (
            OffsetType & offset )
```

Casts into an equivalent plain C++ type.

Some FlatData types can be cast to their equivalent **plain** definition as a regular non-FlatData C++ type. This is a more efficient way to access the data. This function casts, if possible, the member pointed to by the offset argument to an equivalent plain C++ type. Any changes made through the plain type are reflected in the FlatData sample.

Template Parameters

OffsetType	The Offset type

Precondition

plain_cast requires the type to meet the following restrictions:

- The type must be a final struct, or an array or sequence of members of a type that meets these restrictions (including primitive types)
- The type must be defined in a way such that the packing of the plain C++ type doesn't differ from the padding in XCDR2.
- The type may not inherit from another type.
- In addition, the sample must be serialized in the native endianness. For example, if a subscribing application on a little-endian platform receives a sample published from a big-endian system, it is not possible to plain↩ _cast the sample or any of its members, except if the member is a primitive array or sequence of 1-byte elements.

offset.is_cpp_compatible() (p. 1876) indicates if the member meets the requirements. If the type doesn't, this function throws **dds::core::PreconditionNotMetError** (p. 1941).

Parameters

offset	The offset to the member to cast.

Returns

The data that `offset` referred to, cast as a plain C++ type with the same definition. If `offset` is an array or sequence member, the returned pointer represents a C++ array with the same number of elements.

The following table summarizes the possible parameters to this function (assuming that they meet the previous requirements), and the return type in each case:

OffsetType	return type
Offset to a final struct, such as **MyFlatFinalOffset** (p. 1759)	MyFlatFinalPlainHelper* (pointer to a single element). This is a type with the same IDL definition, but without @language_binding(FLAT_DATA).
Offset to an array of final structs, such as **FinalArrayOffset** (p. 1567)<**MyFlatFinalOffset** (p. 1759), N>	MyFlatFinalPlainHelper* (array of N elements)
Offset to a sequence of final structs, such as **SequenceOffset** (p. 2343)<**MyFlatFinalOffset** (p. 1759)>	MyFlatFinalPlainHelper* (array of offset.element_count() elements)
Offset to an array of primitive types, such as **PrimitiveArrayOffset** (p. 1951)<int32_t, N>	int32_t* (array of N elements)
Offset to a sequence of primitive types, such as **PrimitiveSequenceOffset** (p. 1958)<int32_t>	int32_t* (array of offset.element_count() elements)

Due to the performance advantages that plain_cast offers, it is recommended to define FlatData types in a way that their largest member(s) can be plain_cast.

The following example shows how to use plain_cast to cast a **MyFlatFinalOffset** (p. 1759) into the type with the same IDL definition as **MyFlatFinal** (p. 413), but without the @language_binding(FLAT_DATA) annotation:

```
MyFlatMutable *my_sample = ...;
auto my_root = my_sample->root();
auto my_final = my_root.my_final();
// Get the plain C++ type and modify an array
auto my_final_plain = rti::flat::plain_cast(my_final);
my_final_plain->my_primitive(3); // this is using the plain C++ setter
my_final_plain->my_complex_array()[1].x(20); // this is a std::array
// The change to my_complex_array is reflected in the FlatData sample
std::cout « my_final.my_primitive() « std::endl; // 3
std::cout « my_final.my_complex_array().get_element(1).x() « std::endl; // 20
```

This example shows how to use plain_cast to efficiently build a sequence of final elements and a sequence of integers, both members of a mutable type:

```
MyFlatMutableBuilder builder = rti::flat::build_data(writer);
// 1)
//
// Build a sequence of 5 elements with add_n() instead of 5 calls to
// add_next()
rti::flat::FinalSequenceBuilder<MyFlatFinalOffset> seq_builder =
            builder.build_my_final_seq();
seq_builder.add_n(5);
// Finish the member, getting the offset to the member
rti::flat::SequenceOffset<MyFlatFinalOffset> seq_offset = seq_builder.finish();
// Cast it to an array of plain C++ type and initialize it.
// This way to initialize it is more efficient than add_next().
auto seq_elements = rti::flat::plain_cast(seq_offset);
for (int i = 0; i < 5; i++) {
    seq_elements[i].my_primitive(i);
    // ...
}
// 2)
//
// Build a sequence of 1000 integers
auto seq_builder = builder.build_my_primitive_seq();
// make space for 1000 elements, but leave them uninitialized
seq_builder.add_n(1000);
auto seq_offset = seq_builder.finish();
// Initialize the elements:
int32_t *elements = rti::flat::plain_cast(seq_offset);
for (int i = 0; i < 1000; i++) {
    elements[i] = ...;
}
// ... continue building the sample using 'builder'
```

6.117.3.2 plain_cast() [2/2]

```
template<typename OffsetType >
const  flat_type_traits< OffsetType >::plain_type * rti::flat::plain_cast (
             const OffsetType & offset )
```

Const version of plain_cast.

6.118 FlatData Topic-Types

<<***extension***>> *(p. 354)* FlatData Language Binding for IDL topic-types

Modules

- **FlatData Builders**

  A Builder allows creating and initializing variable-size data.
- **FlatData Samples**

  A Sample represents an instance of the IDL topic-type and contains the data in serialized format.
- **FlatData Offsets**

  Offsets provide access to the members of a FlatData Sample.

6.118.1 Detailed Description

<<***extension***>> *(p. 354)* FlatData Language Binding for IDL topic-types

Note

For a complete description of the FlatData language binding and its benefits, and a tutorial, see the "Sending Large Data" chapter in the **RTI Connext User's Manual**.

For buildable **code examples**, see `https://community.rti.com/kb/flatdata-and-zerocopy-examples`.

The FlatData language binding is available in the Traditional C++ API and in the Modern C++ API.

RTI FlatData™ is a language binding for IDL types in which the in-memory representation of a sample matches the wire representation. Therefore, the cost of serialization/deserialization is zero.

To select FlatData as the language binding of a type, annotate it with `@language_binding(FLAT_DATA)` in IDL or apply the attribute `languageBinding="flat_data"` in XML.

There are some restrictions regarding the kinds of types to which the FlatData language binding can be applied.

- For final types, the FlatData language binding can be applied only to fixed-size types. A fixed-size type is a type whose wire representation always has the same size. This includes primitive types, arrays of fixed-size types, and structs containing only members of fixed-size types. Unions are not fixed-size types.
- For mutable types, any member is permitted.

- Extensible types cannot be marked as FlatData.

Final types are more efficient, but more restrictive. In general, a good compromise is to define mutable top-level types, but making sure their largest data members are final.

When a type is marked with the FlatData language binding, its mapping into C++ is different than that of a regular type (plain language binding). Rather than a single C++ class with direct access to its members, for a FlatData type **rtiddsgen** generates the following:

- A **Sample** (p. 412), the data in serialized format
- An **Offset** (p. 414), which allows reading the data members of that type inside a Sample, and modifying them without changing the size
- A **Builder** (p. 409), if the type is mutable, which allows creating a variable-size Sample

For example, for the IDL types **MyFlatFinal** (p. 413) and **MyFlatMutable** (p. 413) **rtiddsgen** generates the following C++ types:

- The Sample types **MyFlatFinal** (p. 413) and **MyFlatMutable** (p. 413)
- The Offset types **MyFlatFinalOffset** (p. 1759) and **MyFlatMutableOffset** (p. 1770).
- The Builder type **MyFlatMutableBuilder** (p. 1763).

6.118.2 Publishing FlatData

The typical way to publish FlatData samples includes the following steps:

- Create a **dds::pub::DataWriter** (p. 1138) as you would for a non-FlatData (plain) topic-type (see **Setting up a DataWriter** (p. 311)).

(For final topic-types such as **MyFlatFinal** (p. 413))

- Obtain a FlatData sample with **dds::pub::DataWriter::get_loan** (p. 1183).
```
MyFlatFinal *foo_sample = writer.get_loan();
```
- Initialize the sample, starting from the **root()** (p. 2276) offset.
```
MyFlatFinalOffset foo_root = foo_sample->root();
foo_root.my_primitive(3);
auto my_complex_offset = foo_root.my_complex();
// ... initialize my_complex_offset
```

(For mutable topic-types such as **MyFlatMutable** (p. 413))

- Obtain a **Builder** (p. 409) to create a variable-size sample with **rti::flat::build_data()** (p. 411).
```
MyFlatMutableBuilder foo_builder = rti::flat::build_data(writer);
```

- Use this builder (and possibly nested member builders) to initialize the members.
```
foo_builder.add_my_primitive(3);
auto my_mutable_builder = foo_builder.build_my_mutable();
// ... build 'my_mutable' (a member with a mutable type)
my_mutable_builder.finish(); // completes a member
auto my_final_offset = foo_builder.add_my_final();
// ... initialize 'my_final' (a member with a final type)
```
- Obtain the completed sample with **MyFlatMutableBuilder::finish_sample()** (p. 1766). After that, the builder is no longer usable, and the sample size cannot change.
```
MyFlatMutable *foo_sample = foo_builder.finish_sample();
```
- Optionally, it is possible to change the values of the sample accessing its root(), as long as the size doesn't change. For example, if the sample was built with a sequence member with two elements, it is possible to modify any of those elements, but it's not possible to add a third element.

(For both final and mutable topic-types)

- Write the sample with **dds::pub::DataWriter::write()** (p. 1148).
```
writer.write(*foo_sample);
```

After write, the DataWriter owns the FlatData sample. This allows avoiding additional copies, the main goal of the Flat↩ Data language binding. This means that the sample cannot be reused. The DataWriter will return it to the sample pool when appropriate, as described in **dds::pub::DataWriter::get_loan** (p. 1183).

6.118.3 Subscribing to FlatData

To subscribe to a topic using a FlatData topic-type:

- Create a **dds::sub::DataReader** (p. 976) normally (see **Setting up a DataReader** (p. 316)).
- Read or take the samples using a loaning operation, such as **dds::sub::DataReader::take** (p. 997) (copying read/take operations cannot be used with FlatData types).
- Access the **root()** (p. 2276) offset, and any of the sample's members from there.

Note that the language binding is a local concept. It is possible to publish with a FlatData topic-type and subscribe to it with a **plain** topic-type with the same (or assignable) definition. It is also possible to use a plain topic-type on the publisher side and subscribe to it using FlatData. The DataWriter or DataReader of the plain topic-type has to use **dds::core::policy::DataRepresentation::xcdr2()** (p. 521) in **dds::core::policy::DataRepresentation** (p. 1113).

6.119 RPC Tutorial

Getting Started with Remote Procedure Call with DDS.

Getting Started with Remote Procedure Call with DDS.

This tutorial will walk you through the steps to define an interface and write the service and client applications:

- **Defining a service interface in IDL** (p. 422)
- **Writing the Service application** (p. 422)
- **Writing the Client application** (p. 424)
- **Advanced interface definition** (p. 424)

6.119.1 Defining a service interface in IDL

The first step is to define a `@service`-annotated `interface` in IDL:

```
@final
struct Coordinates {
    int32 x;
    int32 y;
};
@service
interface RobotControl {
    Coordinates walk_to(Coordinates destination, float speed);
    float get_speed();
};
```

This IDL file defines an interface with two methods:

- `walk_to` receives two arguments and returns a `Coordinates` structure
- `get_speed` takes no arguments and returns a `float`.

These are the methods that we will implement in the Service application, and the methods we will call in the Client application.

To generate the supporting code and example client and service applications, copy the IDL code above into a file called RobotControl.idl and run rtiddsgen as follows (see the `Code Generator User's Manual` for more information):

```
<Connext DDS installation directory>/bin/rtiddsgen -language C++11 -example <your architecture> RobotControl.idl
```

This will generate the following files:

- `RobotControl.hpp`, `RobotControl.cxx`, `RobotControlPlugin.hpp`, `RobotControlPlugin.cxx`. These files contain the supporting types. You should not edit these files.
- `RobotControl_service.cxx` contains the example service application
- `RobotControl_client.cxx` contains the example client application

In the following sections we will modify `RobotControl_service.cxx` and `RobotControl_client.cxx`.

6.119.2 Writing the Service application

`RobotControl_service.cxx` contains an implementation of the `RobotControl` interface:

```
class RobotControlExample : public RobotControl {
public:
    Coordinates walk_to(const Coordinates& destination, float speed) override
    {
        std::cout « "calling walk_to" « std::endl;
        // Implement walk_to function here
        return Coordinates();
    }
    float get_speed() override
    {
        std::cout « "calling get_speed" « std::endl;
        // Implement get_speed function here
        return 0.0f;
    }
};
```

The generated methods simply print a message and return a default value. You can modify it to do something a little more interesting:

```
class RobotControlExample : public RobotControl {
public:
    Coordinates walk_to(const Coordinates& destination, float speed) override
    {
        if (speed_ > 0.0f) {
            std::cout « "already moving" « std::endl;
            return position_; // return previously known position
        }
        if (speed > 100.0f) {
            speed_ = 100; // maximum speed
        } else if (speed <= 0.0f) {
            std::cout « "staying put at position " « position_ « std::endl;
            return position_;
        } else {
            speed_ = speed;
        }
        std::cout « "walking to " « destination « " at a speed of "
                  « speed_ « "...\n";
        // sleep for 0 to 10 seconds, depending on the speed
        std::this_thread::sleep_for(std::chrono::milliseconds(
                10000 - static_cast<int>(100 * speed_)));
        std::cout « "arrived!\n";
        speed_ = 0.0f;
        position_ = destination;
        return destination;
    }
    float get_speed() override
    {
        return speed_;
    }
private:
    Coordinates position_ {0, 0};
    std::atomic<float> speed_ {0.0f};
};
```

The next step is to create a `Server` and a `RobotControlService` that uses an instance of `MyRobotControl`.

```
// Create a DomainParticipant with default Qos. The Service will communicate
// only with Clients that join the same domain_id
dds::domain::DomainParticipant participant(domain_id);
// Create an instance of the service interface
auto service_impl = std::make_shared<RobotControlExample>();
// A server provides the execution environment (a thread pool) for one or
// more services
dds::rpc::ServerParams server_params;
// Use a thread_pool_size > 1 to dispatch incoming function calls in
// parallel
server_params.extensions().thread_pool_size(4);
// Create the server with the server_params configuration
dds::rpc::Server server(server_params);
// Create a service with an instance of the service interface, and attach
// it to a server. The service will wait for remote calls, call the
// service interface to compute the results, and send them back.
dds::rpc::ServiceParams params(participant);
params.service_name("Example RobotControl Service");
RobotControlService service(service_impl, server, params);
std::cout « "RobotControlService running... " « std::endl;
server.run();
```

A Service instance creates the necessary DDS entities (**dds::topic::Topic** (p. 2474), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)) to receive function calls, process them using an interface implementation, and send back the return values. Several Service instances (of the same or different interfaces) can share the same **dds↩ ::domain::DomainParticipant** (p. 1326).

This `RobotControlService` will communicate with clients that run on the same domain id and specify the same service name, `"Example RobotControl Service"`.

The execution of Service instances is managed by a `Server`. A `Server` provides a thread pool for one or more Service instances (of the same or different interfaces). By default, a `Server` creates only one thread, and therefore all the function calls are processed sequentially. In the example, the thread pool size is set to 4. Keep in mind that now the `MyRobotControl` methods can be executed in parallel, so you may need to provide mutual exclusion for shared variables and critical areas.

The RobotControlService is ready to receive remote calls as soon as you instantiate it. Server and RobotControlService are **reference types** (p. 351), and will be destroyed when they're not referenced by any variable. The call to server.run() simply sleeps so that server and service don't go out scope.

6.119.3 Writing the Client application

In RobotControl_client.cxx, we create a RobotControlClient that allows making remote function calls to the service:

```
// Create a DomainParticipant with default Qos
dds::domain::DomainParticipant client_participant(domain_id);
// Create a ClientParams
dds::rpc::ClientParams client_params(client_participant);
// And register the service
client_params.service_name("Example RobotControl Service");
// Create a client and assign the ClientParams
RobotControlClient client(client_params);
// Wait until the service is started
client.wait_for_service();
```

The client is configured with the same service name we used for the service, "Example RobotControl Service". With wait_for_service we wait until the client's readers and writers have matched with the service's own readers and writers. If the client sends a request before we have matched with a service's endpoints, the request will fail and throw an exception.

Next, we're ready to make function calls. Function calls can be synchronous or asynchronous. Synchronous function calls block until a response is received:

```
Coordinates final_position = client.walk_to(Coordinates(150, 200), 85.0f);
std::cout « "Robot walked to " « final_position « std::endl;
```

Asynchronous funcion calls send the request immediately but return a std::future that will contain the result when it's received:

```
std::future<Coordinates> final_position_future =
        client.walk_to_async(Coordinates(10, 0), 40.0f);
std::this_thread::sleep_for(std::chrono::seconds(1));
// The walk_to function may still be running, but we can get the speed
// before we receive the result:
std::cout « "Current speed is " « client.get_speed() « std::endl;
final_position = final_position_future.get();
std::cout « "Robot walked to " « final_position « std::endl;
```

The call to std::future::get() provides the result if it's already available or blocks until it is.

You can configure the maximum wait time for function calls in the ClientParams. By default the wait is infinite. When set to a finite value, operations that do not receive the return value on time throw **dds::core::TimeoutError** (p. 2473).

RobotControlClient is also a **reference type** (p. 351) and will be destroyed automatically when it's no longer referenced by any variable.

6.119.4 Advanced interface definition

In this section we will explore other features you can use to define service interfaces:

- Attributes
- Exceptions
- Out and in/out parameters

An attributes is a data member that generates a getter and a setter function.

Exceptions allow reporting errors in remote function calls.

We will modify our previous IDL to add a `name` attribute for `RobotControl` and an exception for the `walk_to` operation.

```
exception WalkError {
    string<32> message;
};
exception TooFastError {
};
@final
struct Coordinates {
    int32 x;
    int32 y;
};
@service
interface RobotControl {
    Coordinates walk_to(Coordinates destination, float speed)
            raises(WalkError, TooFastError);
    float get_speed();
    attribute string<128> name;
};
```

Now run `rtiddsgen` to update the type files:

```
<Connext DDS installation directory>/bin/rtiddsgen -language C++11 -update
typefiles RobotControl.idl
```

We can now modify `MyRobotControl` to report errors in `walk_to` as exceptions and add the `name` getters and setters:

```
class RobotControlExample : public RobotControl {
public:
    Coordinates walk_to(const Coordinates& destination, float speed) override
    {
        if (speed_ > 0.0f) {
            throw WalkError("Already moving");
        }
        if (speed > 100.0f) {
            throw TooFastError();
        }
        if (speed <= 0.0f) {
            throw WalkError("Invalid speed");
        }
        speed_ = speed;
        std::cout << "walking to " << destination << " at a speed of "
                  << speed_ << "...\n";
        // sleep for 0 to 10 seconds, depending on the speed
        std::this_thread::sleep_for(std::chrono::milliseconds(
                10000 - static_cast<int>(100 * speed_)));
        std::cout << "arrived!\n";
        speed_ = 0.0f;
        position_ = destination;
        return destination;
    }
    float get_speed() override
    {
        return speed_;
    }
    void set_attribute_name(const std::string& name) override
    {
        name_ = name;
    }
    std::string get_attribute_name() override
    {
        return name_;
    }
private:
    std::string name_ {"Unknown"};
    Coordinates position_ {0, 0};
    std::atomic<float> speed_ {0.0f};
};
```

These exceptions are propagated and rethrown in the client:

```
try {
    client.walk_to(Coordinates(5, 20), 105);
```

```
} catch (const TooFastError&) {
    std::cout « "walk_to failed: too fast\n";
} catch (const WalkError& walk_error) {
    std::cout « "walk_to failed: " « walk_error.message() « std::endl;
}
```

In asynchronous calls, exceptions are thrown in `std::future::get()`.

If you need to return more than one value in a method, you can wrap the values in a struct and use that as the return type (as we do in `walk_to`) but you can also specify `out` and `inout` parameters (by default parameters are `in`).

For example, an alternative definition of `walk_to` can use an inout parameter to pass the target position and return the final position, and return a boolean indicating whether the operation succeeded.

```
boolean walk_to(inout Coordinates destination, float speed);
```

This changes how the interfaces are generated. The synchronous interface (`RobotControl`) now receives the `destination` argument by non-const reference so it can be modified. The asynchronous interface (`RobotControlAsync`) returns a future that wraps all the output values: the return type, and all the out and inout parameters.

6.120 Conditions and WaitSets

dds::core::cond::Condition (p. 948) and **dds::core::cond::WaitSet** (p. 2617).

Modules

- **AsyncWaitSet**

  *<<**extension**>> (p. 354) A specialization of **dds::core::cond::WaitSet** (p. 2617) that provides a mechanism to perform the wait asynchronously and uses a thread pool to dispatch the attached active **dds::core::cond::Condition** (p. 948).*

Classes

- class **dds::core::cond::Condition**

  *<<**reference-type**>> (p. 351) Abstract base class of all the conditions*
- class **dds::core::cond::GuardCondition**

  *<<**reference-type**>> (p. 351) A condition whose trigger value is under the control of the application.*
- class **dds::core::cond::StatusCondition**

  *<<**reference-type**>> (p. 351) A condition associated with each **dds::core::Entity** (p. 1520)*
- class **dds::core::cond::WaitSet**

  *<<**reference-type**>> (p. 351) Allows an application to wait until one or more of the attached **Condition** (p. 948) objects have a trigger_value of true or else until the timeout expires.*
- class **rti::core::cond::WaitSetProperty**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Specifies the **dds::core::cond::WaitSet** (p. 2617) behavior for multiple trigger events*

6.120.1 Detailed Description

dds::core::cond::Condition (p. 948) and **dds::core::cond::WaitSet** (p. 2617).

6.121 Time Support

Time and duration types.

Classes

- class **dds::core::Duration**

  <<***value-type***>> (p. *350*) *Represents a time interval*
- class **dds::core::Time**

  <<***extension***>> (p. *354*) *Represents a point in time*

6.121.1 Detailed Description

Time and duration types.

6.122 Exceptions

Classes

- class **dds::core::Exception**

  The abstract base class for all of the DDS exceptions which may be thrown by the API.
- class **dds::core::Error**

  A generic, unspecified ***Error*** (p. *1539*).
- class **dds::core::AlreadyClosedError**

  Indicates that an object has been closed.
- class **dds::core::IllegalOperationError**

  Indicates that an operation was called under improper circumstances.
- class **dds::core::NotAllowedBySecurityError**

  Indicates that an operation on the DDS API fails because the security plugins do not allow it.
- class **dds::core::ImmutablePolicyError**

  Indicates that the application attempted to modify an immutable QoS policy.
- class **dds::core::InconsistentPolicyError**

  Indicates that the application specified a set of QoS policies that are not consistent with each other.
- class **dds::core::InvalidArgumentError**

  Indicates that the application passed an illegal parameter value into an operation.
- class **dds::core::NotEnabledError**

  A ***NotEnabledError*** (p. *1869*) *is thrown when an operation is invoked on a* ***dds::core::Entity*** (p. *1520*) *that is not yet enabled.*
- class **dds::core::OutOfResourcesError**

  Indicates that RTI Connext ran out of the resources needed to complete the operation.
- class **dds::core::PreconditionNotMetError**

  A ***PreconditionNotMetError*** (p. *1941*) *is thrown when a pre-condition for the operation was not met.*
- class **dds::core::TimeoutError**

Indicates that an operation has timed out.

- class **dds::core::UnsupportedError**

  Indicates that the application used an unsupported operation.
- class **dds::core::InvalidDowncastError**

  Indicates that a downcast was incorrect.
- class **dds::core::NullReferenceError**

  Indicates an attempt to access a null object.
- class **rti::core::Result**< **T** >

  A result from an operation that doesn't throw exceptions containing a return code and (if successful) a value.

6.122.1 Detailed Description

6.122.2 Standard Exceptions

Any operation can can throw one of the following "standard exceptions" unless it indicates the opposite:

- **dds::core::Error** (p. 1539)
- **dds::core::IllegalOperationError** (p. 1608)
- **dds::core::AlreadyClosedError** (p. 806)
- **dds::core::InvalidArgumentError** (p. 1619)
- **dds::core::UnsupportedError** (p. 2590)

Operations that throw other exceptions will document them explicitly.

As a general rule constructors may only throw **dds::core::Error** (p. 1539). Destructors never throw, even in case of error.

Note that any operation can throw C++ standard exceptions such as `std::bad_alloc`.

6.122.3 Catching exceptions

The class **dds::core::Exception** (p. 1546) is the abstract base class for all DDS exceptions. All concrete exceptions inherit also from subclasses of `std::exception`, so when you need to catch an exception you have flexibility on the level of detail you need. For example, if you are setting the DomainParticipant QoS:

```
try {
    my_participant.qos(my_participant_qos);
} catch (const dds::core::InconsistentPolicyError& ipe) {
    // catch a specific exception (documented in DomainParticipant::qos())
} catch (const std::exception& ex) {
    // catch any other exception here, including other DDS exceptions
}
```

6.123 QoS Policy Traits

The traits types **policy_id** (p. 1940) and **policy_name** (p. 1941) provide information about a QoS policy.

Classes

- class **dds::core::policy::policy_id**< **Policy** >
  Obtains the QosPolicyId of a QoS Policy.
- class **dds::core::policy::policy_name**< **Policy** >
  Obtains the policy name.

6.123.1 Detailed Description

The traits types **policy_id** (p. 1940) and **policy_name** (p. 1941) provide information about a QoS policy.

6.124 Safe Enumeration

Describes the **safe_enum** (p. 2265) class.

Classes

- class **dds::core::safe_enum**< **def, inner** >
  <<***value-type***>> *(p. 350) Provides a safe, scoped enumeration based on def::type*

6.124.1 Detailed Description

Describes the **safe_enum** (p. 2265) class.

6.125 Status Kinds

Kinds of communication status.

Classes

- class **dds::core::status::StatusMask**
  A std::bitset (list) of statuses.
- class **dds::core::status::InconsistentTopicStatus**
  *Information about the status **dds::core::status::StatusMask::inconsistent_topic()** (p. 2379)*
- class **dds::core::status::SampleLostStatus**
  *Information about the status **dds::core::status::StatusMask::sample_lost()** (p. 2381)*
- class **dds::core::status::SampleRejectedStatus**
  *Information about the status **dds::core::status::StatusMask::sample_rejected()** (p. 2382)*
- class **dds::core::status::LivelinessLostStatus**
  *Information about the status **dds::core::status::StatusMask::liveliness_lost()** (p. 2383)*

- class **dds::core::status::LivelinessChangedStatus**

  *Information about the status **dds::core::status::StatusMask::liveliness_changed()** (p. 2383)*
- class **dds::core::status::OfferedDeadlineMissedStatus**

  *Information about the status **dds::core::status::StatusMask::offered_deadline_missed()** (p. 2379)*
- class **dds::core::status::RequestedDeadlineMissedStatus**

  *Information about the status **dds::core::status::StatusMask::requested_deadline_missed()** (p. 2380)*
- class **dds::core::status::OfferedIncompatibleQosStatus**

  *Information about the status **dds::core::status::StatusMask::offered_incompatible_qos()** (p. 2380)*
- class **dds::core::status::RequestedIncompatibleQosStatus**

  *Information about the status **dds::core::status::StatusMask::requested_incompatible_qos()** (p. 2381)*
- class **dds::core::status::PublicationMatchedStatus**

  *Information about the status **dds::core::status::StatusMask::publication_matched()** (p. 2384)*
- class **dds::core::status::SubscriptionMatchedStatus**

  *Information about the status **dds::core::status::StatusMask::subscription_matched()** (p. 2384)*
- class **rti::core::status::EventCount< IntegerType >**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Encapsulates an event count containing a total count and an incremental count since the last time a status was read.*
- class **rti::core::status::DataReaderCacheStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::datareader_cache()** (p. 2386)*
- class **rti::core::status::DataReaderProtocolStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::datareader_protocol()** (p. 2387)*
- class **rti::core::status::DataWriterCacheStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::datawriter_cache()** (p. 2386)*
- class **rti::core::status::ReliableWriterCacheChangedStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385)*
- class **rti::core::status::ReliableReaderActivityChangedStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::reliable_reader_activity_changed()** (p. 2385)*
- class **rti::core::status::DataWriterProtocolStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::datawriter_protocol()** (p. 2386)*
- class **rti::core::status::DomainParticipantProtocolStatus**
- class **rti::core::status::ServiceRequestAcceptedStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::service_request_accepted()** (p. 2388)*
- class **rti::core::status::InvalidLocalAdvanceNoticeStatus**

  *<<**extension**>> (p. 354) Information about the status **dds::core::status::StatusMask::invalid_local_advance_notice()** (p. 2388)*

6.125.1 Detailed Description

Kinds of communication status.

Entity:

dds::core::Entity (p. 1520)

QoS:

All QoS Policies (p. 508)

Listener:

Listener (p. 1637)

Each concrete **dds::core::Entity** (p. 1520) is associated with a set of Status objects whose value represents the communication status of that entity. Each status value can be accessed with a corresponding method on the **dds::core::Entity** (p. 1520).

When these status values change, the corresponding **dds::core::cond::StatusCondition** (p. 2372) objects are activated and the proper **Listener** (p. 1637) objects are invoked to asynchronously inform the application.

An application is notified of communication status by means of the **Listener** (p. 1637) or the **dds::core::cond::WaitSet** (p. 2617) / **dds::core::cond::Condition** (p. 948) mechanism. The two mechanisms may be combined in the application (e.g., using **dds::core::cond::WaitSet** (p. 2617) (s) / **dds::core::cond::Condition** (p. 948) (s) to access the data and **Listener** (p. 1637) (s) to be warned asynchronously of erroneous communication statuses).

It is likely that the application will choose one or the other mechanism for each particular communication status (not both). However, if both mechanisms are enabled, then the **Listener** (p. 1637) mechanism is used first and then the **dds::core::cond::WaitSet** (p. 2617) objects are signalled.

The statuses may be classified into:

- *read communication statuses*: i.e., those that are related to arrival of data, namely **dds::core::status::StatusMask::data_on_readers()** (p. 2382) and **dds::core::status::StatusMask::data_available()** (p. 2383).

- *plain communication statuses*: i.e., all the others.

Read communication statuses are treated slightly differently than the others because they don't change independently. In other words, at least two changes will appear at the same time (**dds::core::status::StatusMask::data_on_readers()** (p. 2382) and **dds::core::status::StatusMask::data_available()** (p. 2383)) and even several of the last kind may be part of the set. This 'grouping' has to be communicated to the application.

For each plain communication status, there is a corresponding structure to hold the status value. These values contain the information related to the change of status, as well as information related to the statuses themselves (e.g., contains cumulative counts).

"Status Values"

6.125.2 Changes in Status

Associated with each one of an **dds::core::Entity** (p. 1520)'s communication status is a logical `StatusChangedFlag`. This flag indicates whether that particular communication status has changed since the last time the status was read by the application. The way the status changes is slightly different for the Plain Communication Status and the Read Communication status.

"\p StatusChangedFlag indicates if status has changed"

6.125.2.1 Changes in plain communication status

For the plain communication status, the `StatusChangedFlag` flag is initially set to FALSE. It becomes TRUE whenever the plain communication status changes and it is reset to false each time the application accesses the plain communication status via the proper getter operation on the **dds::core::Entity** (p. 1520).

The communication status is also reset to FALSE whenever the associated listener operation is called as the listener implicitly accesses the status which is passed as a parameter to the operation. The fact that the status is reset prior to calling the listener means that if the application calls the getter from inside the listener it will see the status already reset.

An exception to this rule is when the associated listener is the 'nil' listener. The 'nil' listener is treated as a NOOP and the act of calling the 'nil' listener does not reset the communication status.

For example, the value of the `StatusChangedFlag` associated with the **dds::core::status::StatusMask↩ ::requested_deadline_missed()** (p. 2380) will become TRUE each time new deadline occurs (which increases **dds::core::status::RequestedDeadlineMissedStatus::total_count** (p. 2184)). The value changes to FALSE when the application accesses the status via the corresponding **dds::sub::DataReader::requested_deadline_missed_↩ status** (p. 1011) method on the proper **Entity** (p. 1520)

"Changes in \p StatusChangedFlag for plain communication status"

6.125.2.2 Changes in read communication status

For the read communication status, the `StatusChangedFlag` flag is initially set to FALSE. The `Status↩ ChangedFlag` becomes TRUE when either a data-sample arrives or else the **dds::sub::status::ViewState** (p. 2614), **dds::sub::status::SampleState** (p. 2316), or **dds::sub::status::InstanceState** (p. 1615) of any existing sample changes for any reason other than a call to **dds::sub::DataReader::read** (p. 996), **dds::sub::DataReader::take** (p. 997) or their variants. Specifically any of the following events will cause the `StatusChangedFlag` to become TRUE:

- The arrival of new data.
- A change in the **dds::sub::status::InstanceState** (p. 1615) of a contained instance. This can be caused by either:
  - The arrival of the notification that an instance has been disposed by:
    - the **dds::pub::DataWriter** (p. 1138) that owns it if **OWNERSHIP** (p. 538) QoS kind= **dds::core↩ ::policy::OwnershipKind_def::EXCLUSIVE** (p. 1906)
    - or by any **dds::pub::DataWriter** (p. 1138) if **OWNERSHIP** (p. 538) QoS kind= **dds::core::policy::↩ OwnershipKind_def::SHARED** (p. 1906)
  - The loss of liveliness of the **dds::pub::DataWriter** (p. 1138) of an instance for which there is no other **dds↩ ::pub::DataWriter** (p. 1138).
  - The arrival of the notification that an instance has been unregistered by the only **dds::pub::DataWriter** (p. 1138) that is known to be writing the instance.

Depending on the kind of `StatusChangedFlag`, the flag transitions to FALSE again as follows:

- The **dds::core::status::StatusMask::data_available()** (p. 2383) `StatusChangedFlag` becomes FALSE when either the corresponding listener operation (on_data_available) is called or the read or take operation (or their variants) is called on the associated **dds::sub::DataReader** (p. 976).
- The **dds::core::status::StatusMask::data_on_readers()** (p. 2382) `StatusChangedFlag` becomes FALSE when any of the following events occurs:
  - The corresponding listener operation (on_data_on_readers) is called.
  - The on_data_available listener operation is called on any **dds::sub::DataReader** (p. 976) belonging to the **dds::sub::Subscriber** (p. 2410).
  - The read or take operation (or their variants) is called on any **dds::sub::DataReader** (p. 976) belonging to the **dds::sub::Subscriber** (p. 2410).

"Changes in \p StatusChangedFlag for read communication status"

See also

Listener (p. 1637)

dds::core::cond::WaitSet (p. 2617), **dds::core::cond::Condition** (p. 948)

6.126 Supporting Types and Constants

Miscellaneous, general-purpose types and constants.

Classes

- class **dds::core::external**< **T** >

  A managed reference to an object.
- class **dds::core::basic_string**< **CharType, Allocator** >

  <<***value-type***>> *(p. 350) A string convertible to std::string and with similar functionality*
- class **rti::core::bounded_sequence**< **T, MaxLength** >

  <<***value-type***>> *(p. 350) A bounded sequence of elements*
- class **rti::core::ListenerBinder**< **Entity, Listener** >

  <<***reference-type***>> *(p. 351) Automatically manages the association of an Entity and a* ***Listener*** *(p. 1637)*
- class **rti::core::LongDouble**

  <<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350) Encapsulates an IDL* `long double`
- struct **rti::core::xtypes::octet_tag_t**

  <<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350) A special template type to obtain an IDL octet DynamicType from primitive_type*
- class **dds::core::optional**< **T** >

  <<***value-type***>> *(p. 350) Represents an object that may not contain a valid value*
- class **rti::core::optional_value**< **T** >

  <<***extension***>> *(p. 354) Represents a value that can be initialized or not*
- class **rti::core::UnregisterThreadOnExit**

  <<***extension***>> *(p. 354) Utility that calls* ***rti::core::unregister_thread*** *(p. 437) when leaving scope*
- struct **rti::core::qos_print_all_t**

  A tag type that selects the to_string overload that prints all the values of a Qos object.
- class **dds::core::vector**< **T** >

  <<***value-type***>> *(p. 350) A vector convertible to std::vector and with similar functionality*

Typedefs

- using **dds::core::external< T >::shared_ptr** = std::shared_ptr< T >

  The smart pointer that external<T> uses.
- typedef **basic_string**< char, rti::core::memory::OsapiAllocator< char > > **dds::core::string**

  A string convertible to std::string and with similar functionality.
- typedef **basic_string**< DDS_Wchar, rti::core::memory::OsapiAllocator< DDS_Wchar > > **dds::core::wstring**

  An IDL-derived wide string.
- typedef std::vector< uint8_t > **dds::core::ByteSeq**

  A vector of bytes.
- typedef std::vector< std::string > **dds::core::StringSeq**

  A vector of strings.
- typedef std::nullptr_t **dds::core::null_type**

  <<***C++11***>> *(*p. *353) The type of* ***dds::core::null*** *(*p. *438)*
- template<typename T >
  using **omg::types::optional** = **dds::core::optional**< T >

  Optional type according to the IDL4-C++ OMG specification. Alias of ***dds::core::optional*** *(*p. *1879).*
- template<typename T >
  using **omg::types::sequence** = std::vector< T >

  Unbounded sequence type according to the IDL4-C++ OMG specification. Alias of std::vector.
- template<typename T , size_t N>
  using **omg::types::bounded_sequence** = **rti::core::bounded_sequence**< T, N >

  Bounded sequence type according to the IDL4-C++ OMG specification. Alias of ***rti::core::bounded_sequence*** *(*p. *887).*
- using **omg::types::string_view** = std::string_view

  String view type according to the IDL4-C++ OMG specification. Alias of std::string_view when the compiler supports it.
- using **omg::types::wstring_view** = std::wstring_view

  Wstring view type according to the IDL4-C++ OMG specification. Alias of std::wstring_view when the compiler supports it.

Functions

- void **rti::core::unregister_thread** ()

  <<***extension***>> *(*p. *354) Allows you to release thread-specific resources kept by RTI Connext on platforms that do not support automatic thread-specific storage (TSS) cleanup.*

Variables

- const int32_t **dds::core::LENGTH_UNLIMITED** = -1

  A special value indicating an unlimited quantity.
- const **null_type dds::core::null**

  Indicates an empty reference.
- constexpr int32_t **rti::core::length_auto** = -2

  A special value indicating an auto quantity.
- const **qos_print_all_t rti::core::qos_print_all**

  Sentinel value that selects the to_string overload that prints all of the values of a Qos object.

6.126.1 Detailed Description

Miscellaneous, general-purpose types and constants.

6.126.2 Typedef Documentation

6.126.2.1 shared_ptr

```
template<typename T >
using  dds::core::external< T >::shared_ptr = std::shared_ptr<T>
```

The smart pointer that external<T> uses.

6.126.2.2 string

```
typedef  basic_string<char, rti::core::memory::OsapiAllocator<char> >  dds::core::string
```

A string convertible to std::string and with similar functionality.

See also

basic_string (p. 875)

Examples

Foo.hpp.

6.126.2.3 wstring

```
typedef  basic_string<DDS_Wchar, rti::core::memory::OsapiAllocator<DDS_Wchar> >  dds::core↩
::wstring
```

An IDL-derived wide string.

IDL `wstring` maps to this type in the C++ API.

It's character type is `DDS_Wchar`, a portable 4-byte character type.

See also

basic_string (p. 875)

6.126.2.4 ByteSeq

```
typedef std::vector<uint8_t>  dds::core::ByteSeq
```

A vector of bytes.

6.126.2.5 StringSeq

```
typedef std::vector<std::string>  dds::core::StringSeq
```

A vector of strings.

6.126.2.6 null_type

```
typedef std::nullptr_t  dds::core::null_type
```

<<***C++11***>> *(p. 353)* The type of **dds::core::null** (p. 438)

6.126.2.7 optional

```
template<typename T >
using  omg::types::optional = typedef  dds::core::optional<T>
```

Optional type according to the IDL4-C++ OMG specification. Alias of **dds::core::optional** (p. 1879).

6.126.2.8 sequence

```
template<typename T >
using  omg::types::sequence = typedef std::vector<T>
```

Unbounded sequence type according to the IDL4-C++ OMG specification. Alias of std::vector.

6.126.2.9 bounded_sequence

```
template<typename T , size_t N>
using  omg::types::bounded_sequence = typedef  rti::core::bounded_sequence<T, N>
```

Bounded sequence type according to the IDL4-C++ OMG specification. Alias of **rti::core::bounded_sequence** (p. 887).

6.126.2.10 string_view

```
using  omg::types::string_view = typedef std::string_view
```

String view type according to the IDL4-C++ OMG specification. Alias of std::string_view when the compiler supports it.

Examples

Foo.hpp.

6.126.2.11 wstring_view

```
using  omg::types::wstring_view = typedef std::wstring_view
```

Wstring view type according to the IDL4-C++ OMG specification. Alias of std::wstring_view when the compiler supports it.

6.126.3 Function Documentation

6.126.3.1 unregister_thread()

```
void rti::core::unregister_thread ( )
```

<<***extension***>> *(p. 354)* Allows you to release thread-specific resources kept by RTI Connext on platforms that do not support automatic thread-specific storage (TSS) cleanup.

Refer to the `Platform Notes` to see if your platform supports automatic TSS cleanup. It will be mentioned in the 'Thread Configuration' sub-section of the platform you are using.

On platforms that do not support automatic TSS cleanup, you should call this function right before exiting a thread where any DDS API was used. It is safe to call this function even if no DDS API was used, in cases where you are unsure if a DDS API was called or not. It is also safe, although unnecessary, to call this API on platforms that support automatic TSS cleanup.

RTI Connext stores some resources in TSS. For internal threads, these resources are cleaned up by the middleware when the threads exit. For user threads, calling this API will free all the resources related to the specific thread.

The best approach is to call the function during the thread deletion after all the DDS APIs have been called.

See `Unregistering Threads`, in the Core Libraries User's Manual, for the equivalent API call in each of the supported language bindings.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

UnregisterThreadOnExit (p. 2589)

Referenced by **rti::core::UnregisterThreadOnExit::~UnregisterThreadOnExit()**.

6.126.4 Variable Documentation

6.126.4.1 LENGTH_UNLIMITED

```
const int32_t dds::core::LENGTH_UNLIMITED = -1
```

A special value indicating an unlimited quantity.

Referenced by **dds::topic::Topic< T >::discover_any_topic()**, and **dds::topic::Topic< T >::discover_topic_data()**.

6.126.4.2 null

```
const null_type dds::core::null [extern]
```

Indicates an empty reference.

See also

*<<**reference-type**>> (p. 351)*

Referenced by **rti::topic::find_content_filter()**, **dds::sub::DataReader< T >::find_datareader_by_name()**, **dds::sub::DataReader< T >::find_datareader_by_topic_description()**, **dds::sub::DataReader< T >::find_datareader_by_topic_name()**, **dds::pub::DataWriter< T >::find_datawriter_by_name()**, **dds::pub::DataWriter< T >::find_datawriter_by_topic_name()**, and **rti::core::operator<<()**.

6.126.4.3 length_auto

```
constexpr int32_t rti::core::length_auto = -2  [constexpr]
```

A special value indicating an auto quantity.

6.126.4.4 qos_print_all

```
const  qos_print_all_t rti::core::qos_print_all  [extern]
```

Sentinel value that selects the to_string overload that prints all of the values of a Qos object.

See also

rti::sub::qos::to_string

6.127 DynamicType and DynamicData

Describes **dds::core::xtypes::DynamicType** (p. 1504), **dds::core::xtypes::DynamicData** (p. 1461) and related types and functions.

Classes

- struct **dds::core::xtypes::TypeKind_def**
  The definition of TypeKind.
- class **dds::core::xtypes::AliasType**
  *<<**value-type**>> (p. 350) Represents an IDL `typedef`*
- class **dds::core::xtypes::CollectionType**
  *<<**value-type**>> (p. 350) The base class of all collection types*
- class **dds::core::xtypes::UnidimensionalCollectionType**
  *<<**value-type**>> (p. 350) The base class of collection types with only one dimension.*
- class **dds::core::xtypes::SequenceType**
  *<<**value-type**>> (p. 350) Represents an IDL `sequence` type.*
- class **dds::core::xtypes::StringType**
  *<<**value-type**>> (p. 350) Represents an IDL `string` type.*
- class **dds::core::xtypes::WStringType**
  *<<**value-type**>> (p. 350) Represents an IDL `wstring` type.*
- class **dds::core::xtypes::ArrayType**
  *<<**value-type**>> (p. 350) Represents an IDL array type.*
- class **dds::core::xtypes::DynamicData**
  *<<**value-type**>> (p. 350) A data sample of any complex data type, which can be inspected and manipulated reflectively.*
- class **rti::core::xtypes::LoanedDynamicData**
  *<<**move-only-type**>> (p. 353) Gives temporary access to a member of another DynamicData object.*

- class **rti::core::xtypes::DynamicDataInfo**

  Contains information about a DynamicData sample.
- class **rti::core::xtypes::DynamicDataMemberInfo**

  Contains information about a DynamicData member.
- class **rti::core::xtypes::DynamicDataProperty**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Specifies the properties of a DynamicData object*
- class **rti::core::xtypes::DynamicDataTypeSerializationProperty**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Configures aspects of the memory management in the serialization of **dds::core::xtypes::DynamicData** (p. 1461) samples.*
- class **dds::core::xtypes::DynamicType**

  *<<**value-type**>> (p. 350) Represents a runtime type.*
- class **rti::core::xtypes::DynamicTypePrintFormatProperty**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A collection of attributes used to configure how DynamicTypes will be formatted when converted to strings.*
- class **dds::core::xtypes::EnumType**

  *<<**value-type**>> (p. 350) Represents an IDL `enum` type*
- class **dds::core::xtypes::Member**

  *<<**value-type**>> (p. 350) Represents a **StructType** (p. 2400) member*
- class **dds::core::xtypes::UnionMember**

  *<<**value-type**>> (p. 350) Represents a **UnionType** (p. 2584) member*
- class **dds::core::xtypes::EnumMember**

  *<<**value-type**>> (p. 350) Represents a **EnumType** (p. 1535) member*
- class **dds::core::xtypes::PrimitiveType**

  *<<**value-type**>> (p. 350) Represents an IDL primitive type*
- class **dds::core::xtypes::StructType**

  *<<**value-type**>> (p. 350) Represents an IDL `struct` type*
- class **dds::core::xtypes::UnionType**

  *<<**value-type**>> (p. 350) Represents an IDL `union` type*

Typedefs

- typedef **dds::core::safe_enum**< **TypeKind_def** > **dds::core::xtypes::TypeKind**

  The different type kinds.

6.127.1 Detailed Description

Describes **dds::core::xtypes::DynamicType** (p. 1504), **dds::core::xtypes::DynamicData** (p. 1461) and related types and functions.

DynamicType is a mechanism for representing a type at runtime and DynamicData the way to instantiate data samples of a DynamicType and manipulate data reflectively.

6.127.2 Typedef Documentation

6.127.2.1 TypeKind

```
typedef dds::core::safe_enum< TypeKind_def> dds::core::xtypes::TypeKind
```

The different type kinds.

The actual enumeration is in **TypeKind_def** (p. 2571).

See also

DynamicType::kind() (p. 1506)

6.128 Participant Built-in Topics

Builtin topic for accessing information about the DomainParticipants discovered by RTI Connext.

Classes

- class **dds::topic::ParticipantBuiltinTopicData**
  *Entry created when a **dds::domain::DomainParticipant** (p. 1326) is discovered.*

Functions

- std::string **dds::topic::participant_topic_name** ()
  Participant builtin topic name.

6.128.1 Detailed Description

Builtin topic for accessing information about the DomainParticipants discovered by RTI Connext.

6.128.2 Function Documentation

6.128.2.1 participant_topic_name()

```
std::string dds::topic::participant_topic_name ( )
```

Participant builtin topic name.

Topic (p. 2474) name of the builtin **dds::sub::DataReader** (p. 976) for the **dds::topic::ParticipantBuiltinTopicData** (p. 1908) type

6.129 Topic Built-in Topics

Classes

- class **dds::topic::TopicBuiltinTopicData**

  *Entry created when a **dds::topic::Topic** (p. 2474) object is discovered.*

Functions

- std::string **dds::topic::topic_topic_name** ()

  ***Topic** (p. 2474) topic name.*

6.129.1 Detailed Description

Builtin topic for accessing information about the Topics discovered by RTI Connext.

6.129.2 Function Documentation

6.129.2.1 topic_topic_name()

```
std::string dds::topic::topic_topic_name ( )
```

Topic (p. 2474) topic name.

Topic (p. 2474) name of the builtin **dds::sub::DataReader** (p. 976) for the **dds::topic::TopicBuiltinTopicData** (p. 2493) type

6.130 Publication Built-in Topics

Builtin topic for accessing information about the Publications discovered by RTI Connext.

Classes

- class **dds::topic::PublicationBuiltinTopicData**

  *Entry created when a **dds::pub::DataWriter** (p. 1138) is discovered in association with its **dds::pub::Publisher** (p. 1993).*

Functions

- std::string **dds::topic::publication_topic_name** ()

  Publication topic name.

6.130.1 Detailed Description

Builtin topic for accessing information about the Publications discovered by RTI Connext.

6.130.2 Function Documentation

6.130.2.1 publication_topic_name()

```
std::string dds::topic::publication_topic_name ( )
```

Publication topic name.

Topic (p. 2474) name of the builtin **dds::sub::DataReader** (p. 976) for the **dds::topic::PublicationBuiltinTopicData** (p. 1977) type

6.131 Subscription Built-in Topics

Builtin topic for accessing information about the Subscriptions discovered by RTI Connext.

Classes

- class **dds::topic::SubscriptionBuiltinTopicData**

  *Entry created when a **dds::sub::DataReader** (p. 976) is discovered in association with its **dds::sub::Subscriber** (p. 2410).*

Functions

- std::string **dds::topic::subscription_topic_name** ()

  Subscription topic name.

6.131.1 Detailed Description

Builtin topic for accessing information about the Subscriptions discovered by RTI Connext.

6.131.2 Function Documentation

6.131.2.1 subscription_topic_name()

```
std::string dds::topic::subscription_topic_name ( )
```

Subscription topic name.

Topic (p. 2474) name of the builtin **dds::sub::DataReader** (p. 976) for the **dds::topic::SubscriptionBuiltinTopicData** (p. 2428) type

6.132 Topic traits and data-type support

Traits and operations associated to topic-types.

Modules

- **Topic-type serialization and deserialization**

  Provides functions to serialize and deserialize user data types to and from Common Data Representation (CDR) format.

Classes

- struct **dds::topic::is_topic_type< T >**

  *Trait that indicates if a type is suitable to be the type of a **dds::topic::Topic** (p. 2474).*
- struct **dds::topic::topic_type_support< T >**

  Provides convenience operations for a topic-type.
- struct **dds::topic::topic_type_name< T >**

  Provides the name of a topic-type.
- struct **rti::topic::PrintFormatKind_def**

  *The definition of the **dds::core::safe_enum** (p. 2265) PrintFormatKind.*
- class **rti::topic::PrintFormatProperty**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A collection of attributes used to configure how data samples will be formatted when converted to a string.*
- struct **rti::topic::extensibility< TopicType >**

  *<<**extension**>> (p. 354) Indicates the extensibility kind of a topic-type*
- struct **rti::topic::topic_type_disabled_copy< TopicType >**

  *<<**extension**>> (p. 354) Indicates whether a TopicType is uncopyable*
- struct **rti::topic::topic_type_has_external_members< TopicType >**

  *<<**extension**>> (p. 354) Indicates if a topic type contains directly or indirectly IDL external members.*
- struct **rti::topic::dynamic_type< TopicType >**

  Provides a DynamicType that represents an IDL-generated type.

Typedefs

- typedef **dds::core::safe_enum**< **PrintFormatKind_def** > **rti::topic::PrintFormatKind**

  Safe Enumeration *(p. 429) of* ***PrintFormatKind_def*** *(p. 1961) Format kinds available when converting data samples to string representations.*

Functions

- template<typename TopicType >
  std::ostream & **rti::topic::to_string** (std::ostream &out, const TopicType &sample, const **PrintFormatProperty** &print_format= **PrintFormatProperty::Default**())

  Prints a data sample to an output stream.
- template<typename TopicType >
  std::string **rti::topic::to_string** (const TopicType &sample, const **PrintFormatProperty** &print_format= **PrintFormatProperty::Default**())

  Tranforms a data sample into a human-readable string format.
- template<typename TopicType >
  TopicType **rti::topic::from_string** (RTI_STRING_IN str, **PrintFormatKind** format_kind=PrintFormatKind::JSON)

  Creates a data sample from a JSON string representation.

6.132.1 Detailed Description

Traits and operations associated to topic-types.

6.132.2 Typedef Documentation

6.132.2.1 PrintFormatKind

```
typedef dds::core::safe_enum< PrintFormatKind_def> rti::topic::PrintFormatKind
```

Safe Enumeration (p. 429) of **PrintFormatKind_def** (p. 1961) Format kinds available when converting data samples to string representations.

See also

PrintFormatKind_def (p. 1961)

6.132.3 Function Documentation

6.132.3.1 to_string() [1/2]

```
template<typename TopicType >
std::ostream & rti::topic::to_string (
             std::ostream & out,
             const TopicType & sample,
             const  PrintFormatProperty & print_format =  PrintFormatProperty::Default() )
```

Prints a data sample to an output stream.

```
#include <rti/topic/to_string.hpp>
```

Template Parameters

TopicType	A valid topic-type. Valid types are **IDL-generated types** (p. 606) that were generated with typecodes enabled, the **built-in types** (p. 50) and **dds::core::xtypes::DynamicData** (p. 1461).

Parameters

out	The output stream to print to
sample	The sample to print to the output stream
print_format	Properties describing the format with which to print the sample. For example, a sample may be printed in a default format, JSON format or XML format.

Referenced by **dds::topic::qos::TopicQos::operator>>()**.

6.132.3.2 to_string() [2/2]

```
template<typename TopicType >
std::string rti::topic::to_string (
             const TopicType & sample,
             const  PrintFormatProperty & print_format =  PrintFormatProperty::Default() )
```

Tranforms a data sample into a human-readable string format.

```
#include <rti/topic/to_string.hpp>
```

Template Parameters

TopicType	A valid topic-type. Valid types are **IDL-generated types** (p. 606) that were generated with typecodes enabled, the **built-in types** (p. 50) and **dds::core::xtypes::DynamicData** (p. 1461).

Parameters

sample	The sample to get the string representation of
print_format	Properties describing the format with which to print the sample. For example, a sample may be printed in a default format, JSON format or XML format.

6.132.3.3 from_string()

```
template<typename TopicType >
TopicType rti::topic::from_string (
             RTI_STRING_IN str,
              PrintFormatKind format_kind = PrintFormatKind::JSON )
```

Creates a data sample from a JSON string representation.

```
#include <rti/topic/to_string.hpp>
```

Template Parameters

TopicType	An **IDL-generated type** (p. 606), generated with typecodes enabled.

For example:

```
Point p = rti::core::xtypes::from_string<Point>("{ \"x\": 1, \"y\": 2 }");
```

Parameters

str	The JSON string to convert to a sample
format_kind	The only supported value is PrintFormatKind::JSON

Returns

The sample created from the JSON string

See also

dds::core::xtypes::DynamicData::from_string (p. 1489)

References **dds::core::xtypes::DynamicData::from_string()**.

6.133 Activity Context

Add contextual information to log messages.

Classes

- class **rti::config::activity_context::AttributeKindMask**

  The attributes used in the string representation of the Activity Context can be configured through this mask.

Functions

- void **rti::config::activity_context::set_attribute_mask** (**AttributeKindMask** attribute_mask)

  Set the ActivityContextAttributeKindMask of the Activity Context.

6.133.1 Detailed Description

Add contextual information to log messages.

The Activity Context is a group of resources and activities associated with an action such as the creation of an entity.

- A resource is a abstraction of an entity. It can contain attributes such as Topic or domain ID.
- An activity is a general task that the resource is doing, such as "Getting QoS".

Logging context is one of the formats RTI Connext logging infrastructure supports. It is used by default in PrintFormat↩ _def::DEFAULT. It provides information about resources and activities. The activity context is used in two places:

- Logging: activity context is one of the PrintFormat_def DDS logging infrastructure supports. If that format is set every time RTI Connext logs a message, it will contain the contextual information.
- Heap monitoring: every time memory is allocated and heap monitoring is enabled, the string representation of the activity context will be associated with the allocation. This information will be available when taking the snapshot.

For example, in the creation of a DataWriter, the activity context will provide information about:

- Resource: the Publisher creating the DataWriter. Attributes of the publisher will be GUID, kind, name and domain ID.
- Activity: entity creation. It will have two parameters, the entity kind and the Topic. In the example below, these are "Writer" and "TestTopic".

The string representation of the above activity context would be:

```
[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000088{Entity=Pu,Name=TestPublisher,Domain=1}|CREATE Writer WITH TOPIC
       TestTopic]
```

Another example could be when a DataWriter writes a sample. The activity context will provide information about:

- Resource: the DataWriter writing the sample. The attributes of the DataWriter will be GUID, name, kind, Topic, data type, and the domain ID.
- Activity will be "write a sample".

The string representation of the activity context will be:

```
[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]
```

6.133.2 Function Documentation

6.133.2.1 set_attribute_mask()

```
void rti::config::activity_context::set_attribute_mask (
              AttributeKindMask attribute_mask )
```

Set the ActivityContextAttributeKindMask of the Activity Context.

Set the **AttributeKindMask** (p. 863) of the Activity Context.

6.134 Compliance Configuration

APIs to configure compliance with certain standard specifications.

Classes

- class **rti::config::compliance::XTypesMask**

  This mask configures different aspects of the Extended CDR encoding (XCDR and XCDR2).
- class **rti::config::compliance::RpcMask**

  XTypes compliance mask.

Functions

- **XTypesMask rti::config::compliance::get_xtypes_mask** ()

  Gets the XTypes compliance mask value for the application.
- void **rti::config::compliance::set_xtypes_mask** (**XTypesMask** mask)

  Sets the level of compliance of the application with the XTypes specification.
- void **rti::config::compliance::load_compliance_masks** ()

  Load the compliance masks from the environment variables.
- **RpcMask rti::config::compliance::get_rpc_mask** ()

  Gets the RPC compliance mask value for the application.
- void **rti::config::compliance::set_rpc_mask** (**RpcMask** mask)

  Sets the level of compliance of the application with the Rpc specification.

6.134.1 Detailed Description

APIs to configure compliance with certain standard specifications.

6.134.2 Function Documentation

6.134.2.1 get_xtypes_mask()

```
XTypesMask rti::config::compliance::get_xtypes_mask ( )
```

Gets the XTypes compliance mask value for the application.

```
#include <rti/config/Compliance.hpp>
```

Note

This is a standalone function in the namespace rti::config::compliance

The **rti::config::compliance::XTypesMask** (p. 2666) configures different aspects of the Extended CDR encoding (XCDR and XCDR2) for the application.

Returns

The **rti::config::compliance::XTypesMask** (p. 2666) value.

See also

rti::config::compliance::XTypesMask (p. 2666)

rti::config::compliance::get_xtypes_mask() (p. 451)

rti::config::compliance::load_compliance_masks() (p. 453)

6.134.2.2 set_xtypes_mask()

```
void rti::config::compliance::set_xtypes_mask (
            XTypesMask mask )
```

Sets the level of compliance of the application with the XTypes specification.

```
#include <rti/config/Compliance.hpp>
```

Note

This is a standalone function in the namespace rti::config::compliance

The **rti::config::compliance::XTypesMask** (p. 2666) configures different aspects of the Extended CDR encoding (XCDR and XCDR2) for the application.

Once you call this function the value of the mask will apply to all the entities created after the call. It won't apply to entities created before the call.

In general, applications should only set the mask once, before executing any other RTI Connext API. The mask can also be set using the environment variable **NDDS_XTYPES_COMPLIANCE_MASK**.

Parameters

mask	<<***in***>> *(*p. *355)* Compliance mask.

See also

rti::config::compliance::XTypesMask (p. 2666)

rti::config::compliance::get_xtypes_mask() (p. 451)

rti::config::compliance::load_compliance_masks() (p. 453)

6.134.2.3 load_compliance_masks()

```
void rti::config::compliance::load_compliance_masks ( )
```

Load the compliance masks from the environment variables.

```
#include <rti/config/Compliance.hpp>
```

Note

This is a standalone function in the namespace rti::config::compliance

This function loads the XTypes and RPC compliance masks from the environment, which can be set using the environment variables **NDDS_XTYPES_COMPLIANCE_MASK** and **NDDS_RPC_COMPLIANCE_MASK**.

See also

rti::config::compliance::XTypesMask (p. 2666)

rti::config::compliance::set_xtypes_mask() (p. 452)

rti::config::compliance::get_xtypes_mask() (p. 451)

rti::config::compliance::RpcMask (p. 2218)

rti::config::compliance::set_rpc_mask() (p. 454)

rti::config::compliance::get_rpc_mask() (p. 453)

6.134.2.4 get_rpc_mask()

```
RpcMask rti::config::compliance::get_rpc_mask ( )
```

Gets the RPC compliance mask value for the application.

```
#include <rti/config/Compliance.hpp>
```

Note

This is a standalone function in the namespace rti::config::compliance

The **rti::config::compliance::RpcMask** (p. 2218) configures different aspects of RPC communication.

Returns

The **rti::config::compliance::RpcMask** (p. 2218) value.

See also

rti::config::compliance::RpcMask (p. 2218)

rti::config::compliance::get_rpc_mask() (p. 453)

rti::config::compliance::load_compliance_masks() (p. 453)

6.134.2.5 set_rpc_mask()

```
void rti::config::compliance::set_rpc_mask (
            RpcMask mask )
```

Sets the level of compliance of the application with the Rpc specification.

```
#include <rti/config/Compliance.hpp>
```

Note

This is a standalone function in the namespace rti::config::compliance

The **rti::config::compliance::RpcMask** (p. 2218) configures different aspects of RPC communication.

Once you call this function the value of the mask will apply to all the entities created after the call. It won't apply to entities created before the call.

In general, applications should only set the mask once, before executing any other RTI Connext API. The mask can also be set using the environment variable **NDDS_RPC_COMPLIANCE_MASK**.

Parameters

mask	<<***in***>> *(*p. *355)* Compliance mask.

See also

rti::config::compliance::RpcMask (p. 2218)

rti::config::compliance::get_rpc_mask() (p. 453)

rti::config::compliance::load_compliance_masks() (p. 453)

6.135 Logging

Configure how much debugging information is reported during runtime and where it is logged.

Modules

- **Activity Context**

  Add contextual information to log messages.

Classes

- struct **rti::config::LogLevel_def**

  The definition of the ***dds::core::safe_enum*** *(*p. *2265) LogLevel;.*
- struct **rti::config::Verbosity_def**

  The definition of the ***dds::core::safe_enum*** *(*p. *2265) Verbosity.*
- struct **rti::config::LogCategory_def**

  The definition of the ***dds::core::safe_enum*** *(*p. *2265) LogCategory.*
- struct **rti::config::PrintFormat_def**

  The definition of the ***dds::core::safe_enum*** *(*p. *2265) PrintFormat.*
- struct **rti::config::LogMessage**

  A log message's input parameter to an output handler, including the text and additional information.
- class **rti::config::Logger**

  The singleton type used to configure RTI Connext logging.
- class **rti::config::ScopedLoggerVerbosity**

  Changes the logger verbosity temporarily during the scope of a variable.

Typedefs

- typedef **dds::core::safe_enum**< **LogLevel_def** > **rti::config::LogLevel**

  The log level at which RTI Connext diagnostic information is logged.
- typedef **dds::core::safe_enum**< **Verbosity_def** > **rti::config::Verbosity**

  The verbosities at which RTI Connext diagnostic information is logged.
- typedef **dds::core::safe_enum**< **LogCategory_def** > **rti::config::LogCategory**

  Categories of logged messages.
- typedef **dds::core::safe_enum**< **PrintFormat_def** > **rti::config::PrintFormat**

  The format used to output RTI Connext diagnostic information.

Enumerations

- enum class **rti::config::SyslogLevel** {
  rti::config::SyslogLevel::emergency ,
  rti::config::SyslogLevel::alert ,
  rti::config::SyslogLevel::critical ,
  rti::config::SyslogLevel::error ,
  rti::config::SyslogLevel::warning ,
  rti::config::SyslogLevel::notice ,
  rti::config::SyslogLevel::informational ,
  rti::config::SyslogLevel::debug }
- enum class **rti::config::SyslogVerbosity** {
  rti::config::SyslogVerbosity::silent ,
  rti::config::SyslogVerbosity::emergency ,
  rti::config::SyslogVerbosity::alert ,
  rti::config::SyslogVerbosity::critical ,
  rti::config::SyslogVerbosity::error ,
  rti::config::SyslogVerbosity::warning ,
  rti::config::SyslogVerbosity::notice ,
  rti::config::SyslogVerbosity::informational ,
  rti::config::SyslogVerbosity::debug }
- enum class **rti::config::LogFacility** {
  rti::config::LogFacility::user ,
  rti::config::LogFacility::security_event ,
  rti::config::LogFacility::service ,
  rti::config::LogFacility::middleware }

6.135.1 Detailed Description

Configure how much debugging information is reported during runtime and where it is logged.

6.135.2 Typedef Documentation

6.135.2.1 LogLevel

```
typedef dds::core::safe_enum< LogLevel_def> rti::config::LogLevel
```

The log level at which RTI Connext diagnostic information is logged.

See also

LogLevel_def (p. 1692)

6.135.2.2 Verbosity

typedef **dds::core::safe_enum**< **Verbosity_def**> **rti::config::Verbosity**

The verbosities at which RTI Connext diagnostic information is logged.

See also

Verbosity_def (p. 2613)

6.135.2.3 LogCategory

typedef **dds::core::safe_enum**< **LogCategory_def**> **rti::config::LogCategory**

Categories of logged messages.

See also

LogCategory_def (p. 1682)

6.135.2.4 PrintFormat

typedef **dds::core::safe_enum**< **PrintFormat_def**> **rti::config::PrintFormat**

The format used to output RTI Connext diagnostic information.

See also

PrintFormat_def (p. 1960)

6.135.3 Enumeration Type Documentation

6.135.3.1 SyslogLevel

enum class **rti::config::SyslogLevel** [strong]

Syslog level category assigned to RTI Connext log messages. See `Syslog Level and Verbosity Mapping`, in the Core Libraries User's Manual, for more information.

Enumerator

emergency	System is unusable. Maps to **rti::config::LogLevel_def::FATAL_ERROR** (p. 1693).
alert	Should be corrected immediately. Maps to **rti::config::LogLevel_def::EXCEPTION** (p. 1693).
critical	Critical conditions. Maps to **rti::config::LogLevel_def::EXCEPTION** (p. 1693).
error	Error conditions. Maps to **rti::config::LogLevel_def::EXCEPTION** (p. 1693).
warning	May indicate that an error will occur if action is not taken. Maps to **rti::config::LogLevel_def::WARNING** (p. 1693).
notice	Events that are unusual, but not error conditions. Maps to **rti::config::LogLevel_def::STATUS_LOCAL** (p. 1693).
informational	Normal operational messages that require no action. Maps to **rti::config::LogLevel_def::STATUS_LOCAL** (p. 1693).
debug	Information useful to developers for debugging the application. Maps to **rti::config::LogLevel_def::STATUS_ALL** (p. 1693).

6.135.3.2 SyslogVerbosity

`enum class` **`rti::config::SyslogVerbosity`** `[strong]`

The Syslog verbosities at which RTI Connext diagnostic information is logged. These Syslog verbosities are mapped to **rti::config::Verbosity_def** (p. 2613). See `Syslog Level and Verbosity Mapping`, in the Core Libraries User's Manual, for more information.

Enumerator

silent	No messages will be logged. (lowest verbosity) Equivalent to **rti::config::Verbosity_def::silent** (p. 2614).
emergency	Emergency, critical, alert, and error messages will be logged. This Syslog verbosity level is translated to **rti::config::Verbosity_def::exception** (p. 2614) when interpreted by RTI Connext. (That's why this level actually logs more than just emergency messages.)
alert	Emergency, critical, and alert messages will be logged. This Syslog verbosity level is translated to **rti::config::Verbosity_def::exception** (p. 2614) when interpreted by RTI Connext. (That's why this level actually logs more than just emergency, critical and alert messages.)
critical	Emergency, critical, alert, and error messages will be logged. This Syslog verbosity level is translated to **rti::config::Verbosity_def::exception** (p. 2614) when interpreted by RTI Connext. (That's why this level actually logs more than just emergency and critical messages.)
error	Emergency, critical, alert, and error messages will be logged. This Syslog verbosity level is translated to **rti::config::Verbosity_def::exception** (p. 2614) when interpreted by RTI Connext.
warning	Emergency, critical, alert, error, and warning messages will be logged. This Syslog verbosity level is translated to **rti::config::Verbosity_def::warning** (p. 2614) when interpreted by RTI Connext.
notice	Emergency, critical, alert, error, warning, notice and informational messages will be logged. This Syslog verbosity level is translated to **rti::config::Verbosity_def::status_remote** (p. 2614) when interpreted by RTI Connext. (That's why this level actually logs more than just emergency, critical, alert, error, warning and notice messages.)
informational	Emergency, critical, alert, error, warning, notice, and informational messages will be logged. This Syslog verbosity level is translated to **rti::config::Verbosity_def::status_remote** (p. 2614) when interpreted by RTI Connext.
debug	All messages will be logged. This Syslog verbosity level is translated to **rti::config::Verbosity_def::status_all** (p. 2614) when interpreted by RTI Connext.

6.135.3.3 LogFacility

enum class **rti::config::LogFacility** [strong]

A number that identifies the source of a log message.

In the Syslog Protocol, the Facility is a numerical code that represents the machine process that created a Syslog event. RTI Connext uses the facility to represent the source of a given log message.

Enumerator

user	A log message produced by the APIs that log user messages in **rti::config::Logger** (p. 1683) (e.g, **rti::config::Logger::emergency** (p. 1689)). Numerical code: 1
security_event	A security-related message logged by the RTI Security Plugins Logging Plugin. Numerical code: 10
service	A log message produced by an Infrastructure Service, such as Routing Service. Infrastructure Services operate using the Core Libraries. The messages that the Core Libraries create are categorized under the MIDDLEWARE facility. Log messages that are directly generated by the Infrastructure Service itself are marked with the SERVICE facility. Numerical code: 22
middleware	A log message produced by RTI Connext Core Libraries. Numerical code: 23

6.136 Version

Retrieve information for the RTI Connext product, the core library, and the C, C++ or Java libraries.

Classes

- class **rti::config::LibraryVersion**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) The version of a single library shipped as part of an RTI Connext distribution.*

Functions

- XMQCPP2DllExport **rti::core::ProductVersion rti::config::request_reply_api_version** ()

  Get the version of the C++ API library.
- XMQCPP2DllExport std::string **rti::config::request_reply_build_number** ()

  Get the build number of the C++ API library.
- **LibraryVersion rti::config::core_version** ()

  Get the version of the core library.
- std::string **rti::config::core_build_number** ()

  Get the build number of the core library.

- **LibraryVersion rti::config::c_api_version** ()

  Get the version of the C API library.
- std::string **rti::config::c_build_number** ()

  Get the build number of the C API library.
- **LibraryVersion rti::config::cpp_api_version** ()

  Get the version of the C++ API library.
- std::string **rti::config::cpp_build_number** ()

  Get the build number of the C++ API library.
- **rti::core::ProductVersion rti::config::product_version** ()

  Get the RTI Connext product version.

6.136.1 Detailed Description

Retrieve information for the RTI Connext product, the core library, and the C, C++ or Java libraries.

There are three ways to obtain version information: looking at the revision files, using Visual Studio or the command line, or programmatically at run time. This HTML documentation includes a reference for consulting the APIs that allow you to get version information programmatically. For more information see the RTI Connext DDS Core Libraries User's Manual.

The version information includes four fields:

- Major product version.
- Minor product version.
- Release letter for product version.
- Revision number of product.

6.136.2 Function Documentation

6.136.2.1 request_reply_api_version()

```
XMQCPP2DllExport  rti::core::ProductVersion rti::config::request_reply_api_version ( )
```

Get the version of the C++ API library.

6.136.2.2 request_reply_build_number()

```
XMQCPP2DllExport std::string rti::config::request_reply_build_number ( )
```

Get the build number of the C++ API library.

6.136.2.3 core_version()

```
LibraryVersion rti::config::core_version ( )
```

Get the version of the core library.

6.136.2.4 core_build_number()

```
std::string rti::config::core_build_number ( )
```

Get the build number of the core library.

6.136.2.5 c_api_version()

```
LibraryVersion rti::config::c_api_version ( )
```

Get the version of the C API library.

6.136.2.6 c_build_number()

```
std::string rti::config::c_build_number ( )
```

Get the build number of the C API library.

6.136.2.7 cpp_api_version()

```
LibraryVersion rti::config::cpp_api_version ( )
```

Get the version of the C++ API library.

6.136.2.8 cpp_build_number()

```
std::string rti::config::cpp_build_number ( )
```

Get the build number of the C++ API library.

6.136.2.9 product_version()

```
rti::core::ProductVersion rti::config::product_version ( )
```

Get the RTI Connext product version.

6.137 Builtin Qos Profiles

<<***extension***>> *(p. 354)* QoS libraries and profiles that are automatically built into RTI Connext.

Functions

- std::string **rti::core::builtin_profiles::qos_lib::library_name** ()
  A library of non-experimental QoS profiles.
- std::string **rti::core::builtin_profiles::qos_lib::baseline** ()
  The most up-to-date QoS default values.
- std::string **rti::core::builtin_profiles::qos_lib::baseline_5_0_0** ()
  The QoS default values for version 5.0.0.
- std::string **rti::core::builtin_profiles::qos_lib::baseline_5_1_0** ()
  The QoS default values for version 5.1.0.
- std::string **rti::core::builtin_profiles::qos_lib::baseline_5_2_0** ()
  The QoS default values for version 5.2.0.
- std::string **rti::core::builtin_profiles::qos_lib::baseline_5_3_0** ()
  The QoS default values for version 5.3.0.
- std::string **rti::core::builtin_profiles::qos_lib::baseline_6_0_0** ()
  The QoS default values for version 6.0.0.
- std::string **rti::core::builtin_profiles::qos_lib::baseline_6_1_0** ()
  The QoS default values for version 6.1.0.
- std::string **rti::core::builtin_profiles::qos_lib::baseline_7_0_0** ()
  The QoS default values for version 7.0.0.
- std::string **rti::core::builtin_profiles::qos_lib::baseline_7_1_0** ()
  The QoS default values for version 7.1.0.
- std::string **rti::core::builtin_profiles::qos_lib::generic_common** ()
  A common Participant base profile.
- std::string **rti::core::builtin_profiles::qos_lib::generic_monitoring_common** ()
  Enables RTI Monitoring Library.
- std::string **rti::core::builtin_profiles::qos_lib::generic_connext_micro_compatibility** ()
  Sets the values necessary to communicate with RTI Connext Micro.
- std::string **rti::core::builtin_profiles::qos_lib::generic_connext_micro_compatibility_2_4_9** ()
  Sets the values necessary to communicate with RTI Connext Micro versions 2.4.4 through at least 2.4.9.
- std::string **rti::core::builtin_profiles::qos_lib::generic_connext_micro_compatibility_2_4_3** ()
  Sets the values necessary to communicate with RTI Connext Micro versions 2.4.3 and ealier.
- std::string **rti::core::builtin_profiles::qos_lib::generic_other_dds_vendor_compatibility** ()
  Sets the values necessary to interoperate with other DDS vendors.
- std::string **rti::core::builtin_profiles::qos_lib::generic_510_transport_compatibility** ()

Sets the values necessary to interoperate with RTI Connext 5.1.0 using the UDPv6 and/or SHMEM transports.

- std::string **rti::core::builtin_profiles::qos_lib::generic_security** ()

  Loads the DDS Secure builtin plugins.
- std::string **rti::core::builtin_profiles::qos_lib::generic_strict_reliable** ()

  Enables strict reliability.
- std::string **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable** ()

  Enables keep-last reliability.
- std::string **rti::core::builtin_profiles::qos_lib::generic_best_effort** ()

  Enables best-effort reliability kind.
- std::string **rti::core::builtin_profiles::qos_lib::generic_strict_reliable_high_throughput** ()

  A profile that can be used to achieve high throughput.
- std::string **rti::core::builtin_profiles::qos_lib::generic_strict_reliable_low_latency** ()

  A profile that can be used to achieve low latency.
- std::string **rti::core::builtin_profiles::qos_lib::generic_participant_large_data** ()

  A common Participant base profile to facilitate sending large data.
- std::string **rti::core::builtin_profiles::qos_lib::generic_participant_large_data_monitoring** ()

  Configures Participants for large data and monitoring.
- std::string **rti::core::builtin_profiles::qos_lib::generic_strict_reliable_large_data** ()

  Configures endpoints for sending large data with strict reliability.
- std::string **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_large_data** ()

  Configures endpoints for sending large data with keep-last reliability.
- std::string **rti::core::builtin_profiles::qos_lib::generic_strict_reliable_large_data_fast_flow** ()

  Configures strictly reliable communication for large data with a fast flow controller.
- std::string **rti::core::builtin_profiles::qos_lib::generic_strict_reliable_large_data_medium_flow** ()

  Configures strictly reliable communication for large data with a medium flow controller.
- std::string **rti::core::builtin_profiles::qos_lib::generic_strict_reliable_large_data_slow_flow** ()

  Configures strictly reliable communication for large data with a slow flow controller.
- std::string **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_large_data_fast_flow** ()

  Configures keep-last reliable communication for large data with a fast flow controller.
- std::string **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_large_data_medium_flow** ()

  Configures keep-last reliable communication for large data with a medium flow controller.
- std::string **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_large_data_slow_flow** ()

  Configures keep-last reliable communication for large data with a slow flow controller.
- std::string **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_transient_local** ()

  Persists the samples of a DataWriter as long as the entity exists.
- std::string **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_transient** ()

  Persists samples using RTI Persistence Service.
- std::string **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_persistent** ()

  Persists samples in permanent storage, like a disk, using RTI Persistence Service.
- std::string **rti::core::builtin_profiles::qos_lib::generic_auto_tuning** ()

  Enables the Turbo Mode batching and Auto Throttle experimental features.
- std::string **rti::core::builtin_profiles::qos_lib::generic_minimal_memory_footprint** ()

  Uses a set of QoS which reduces the memory footprint of the application.
- std::string **rti::core::builtin_profiles::qos_lib::generic_monitoring2** ()

  The default QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use.
- std::string **rti::core::builtin_profiles::qos_lib::generic_monitoring2_wan** ()

The QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use to communicate over WAN with a Collector Service.

- std::string **rti::core::builtin_profiles::qos_lib::pattern_rpc** ()

  This profile contains the default RPC and Request-Reply QoS settings.
- std::string **rti::core::builtin_profiles::qos_lib::pattern_periodic_data** ()

  Used for applications that expect periodic data.
- std::string **rti::core::builtin_profiles::qos_lib::pattern_streaming** ()

  Used for applications that stream data.
- std::string **rti::core::builtin_profiles::qos_lib::pattern_reliable_streaming** ()

  Used for applications that stream data and require reliable communication.
- std::string **rti::core::builtin_profiles::qos_lib::pattern_event** ()

  Used for applications that handle events.
- std::string **rti::core::builtin_profiles::qos_lib::pattern_alarm_event** ()

  Used for applications that handle alarm events.
- std::string **rti::core::builtin_profiles::qos_lib::pattern_status** ()

  Used for applications whose samples represent statuses.
- std::string **rti::core::builtin_profiles::qos_lib::pattern_alarm_status** ()

  Used for applications in which samples represent alarm statuses.
- std::string **rti::core::builtin_profiles::qos_lib::pattern_last_value_cache** ()

  Used for applications that only need the last published value.
- std::string **rti::core::builtin_profiles::qos_lib_exp::library_name** ()

  A library of experimental QoS profiles.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable** ()

  Enables strict reliability.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable** ()

  Enables keep-last reliability.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_best_effort** ()

  Enables best-effort reliability kind.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable_high_throughput** ()

  A profile that can be used to achieve high throughput.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable_low_latency** ()

  A profile that can be used to achieve low latency.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_participant_large_data** ()

  A common Participant base profile to facilitate sending large data.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_participant_large_data_monitoring** ()

  Configures Participants for large data and monitoring.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable_large_data** ()

  Configures endpoints for sending large data with strict reliability.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_large_data** ()

  Configures endpoints for sending large data with keep-last reliability.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable_large_data_fast_flow** ()

  Configures strictly reliable communication for large data with a fast flow controller.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable_large_data_medium_flow** ()

  Configures strictly reliable communication for large data with a medium flow controller.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable_large_data_slow_flow** ()

  Configures strictly reliable communication for large data with a slow flow controller.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_large_data_fast_flow** ()

Configures keep-last reliable communication for large data with a fast flow controller.

- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_large_data_medium_flow** ()

  Configures keep-last reliable communication for large data with a medium flow controller.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_large_data_slow_flow** ()

  Configures keep-last reliable communication for large data with a slow flow controller.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_transient_local** ()

  Persists the samples of a DataWriter as long as the entity exists.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_transient** ()

  Persists samples using RTI Persistence Service.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_persistent** ()

  Persists samples in permanent storage, like a disk, using RTI Persistence Service.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_auto_tuning** ()

  Enables the Turbo Mode batching and Auto Throttle experimental features.
- std::string **rti::core::builtin_profiles::qos_lib_exp::generic_minimal_memory_footprint** ()

  Uses a set of QoS which reduces the memory footprint of the application.
- std::string **rti::core::builtin_profiles::qos_lib_exp::pattern_periodic_data** ()

  Used for applications that expect periodic data.
- std::string **rti::core::builtin_profiles::qos_lib_exp::pattern_streaming** ()

  Used for applications that stream data.
- std::string **rti::core::builtin_profiles::qos_lib_exp::pattern_reliable_streaming** ()

  Used for applications that stream data and require reliable communication.
- std::string **rti::core::builtin_profiles::qos_lib_exp::pattern_event** ()

  Used for applications that handle events.
- std::string **rti::core::builtin_profiles::qos_lib_exp::pattern_alarm_event** ()

  Used for applications that handle alarm events.
- std::string **rti::core::builtin_profiles::qos_lib_exp::pattern_status** ()

  Used for applications whose samples represent statuses.
- std::string **rti::core::builtin_profiles::qos_lib_exp::pattern_alarm_status** ()

  Used for applications in which samples represent alarm statuses.
- std::string **rti::core::builtin_profiles::qos_lib_exp::pattern_last_value_cache** ()

  Used for applications that only need the last published value.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::library_name** ()

  A library of QoS Snippets.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_common** ()

  QoS Snippet that configures the reliability protocol with a common configuration.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_keep_all** ()

  QoS Snippet that configures the reliability protocol for KEEP_ALL.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_keep_last** ()

  QoS Snippet that configures the reliability protocol for KEEP_LAST.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_high_rate** ()

  QoS Snippet that configures the reliability protocol for sending data at a high rate.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_low_latency** ()

  QoS Snippet that configures the reliability protocol for sending data at low latency.

- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_large_↩ data** ()

  QoS Snippet that configures the reliability protocol for large data.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_↩ dynamicmemalloc** ()

  Configures the DataWriter and DataReader caches to use dynamic memory allocation for Large Data samples.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_discovery_common** ()

  QoS Snippet that optimizes discovery with a common configuration.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_discovery_participant_↩ compact** ()

  QoS Snippet that optimizes the Participant QoS to send less discovery information.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_discovery_endpoint_fast** ()

  QoS Snippet that optimizes the Endpoint Discovery to be faster.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_transport_large_buffers** ()

  QoS Snippet that increases the Participant default buffer that shmem and udpv4 use.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_reliability_reliable** ()

  QoS Snippet that sets RELIABILITY QoS to RELIABLE.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_reliability_best_effort** ()

  QoS Snippet that sets RELIABILITY QoS to BEST_EFFORT.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_history_keep_last_1** ()

  QoS Snippet that sets HISTORY QoS to KEEP_LAST kind with depth 1.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_history_keep_all** ()

  QoS Snippet that sets HISTORY QosPolicy to KEEP_ALL kind.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_publish_mode_asynchronous** ()

  QoS Snippet that sets PUBLISH_MODE QosPolicy to ASYNCHRONOUS kind.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_durability_transient_local** ()

  QoS Snippet that sets DURABILITY QosPolicy to TRANSIENT_LOCAL kind.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_durability_transient** ()

  QoS Snippet that sets DURABILITY QosPolicy to TRANSIENT kind.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_durability_persistent** ()

  QoS Snippet that sets DURABILITY QosPolicy to PERSISTENT kind.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_batching_enable** ()

  QoS Snippet that sets BATCH QosPolicy to true.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_flow_controller_838mbps** ()

  QoS Snippet that configures and set a FlowController of 838 Mbps.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_flow_controller_209mbps** ()

  QoS Snippet that configures and sets a FlowController of 209 Mbps.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_flow_controller_52mbps** ()

  QoS Snippet that configures and sets a FlowController of 52 Mbps.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_auto_tuning_enable** ()

  QoS Snippet that enables auto_throttle and turbo_mode to true.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_monitoring_enable** ()

  QoS Snippet that enables the use of the RTI Monitoring Library.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_monitoring2_enable** ()

  QoS Snippet that enables the use of the RTI Monitoring Library 2.0.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_security_enable** ()

QoS Snippet that enables security using the Builtin Security Plugins.

- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_topic_query_enable** ()

  QoS Snippet that enables Topic Query.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_type_support_v1_enable** ()

  QoS Snippet that enables TypeObject V1 propagation.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_type_support_v1_disable** ()

  QoS Snippet that disables TypeObject V1 propagation.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_type_support_v2_enable** ()

  QoS Snippet that enables TypeObject V2 propagation.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_type_support_v2_disable** ()

  QoS Snippet that disables TypeObject V2 propagation.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_transport_tcp_lan_client** ()

  QoS Snippet that configures a TCP LAN Client over DDS.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_transport_tcp_wan_symmetric_client** ()

  QoS Snippet that configures a symmetric WAN TCP Client over DDS.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_transport_tcp_wan_asymmetric_server** ()

  QoS Snippet that an asymmetric WAN TCP Server over DDS.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_transport_tcp_wan_asymmetric_client** ()

  QoS Snippet that configures an asymmetric WAN TCP Client over DDS.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_transport_udp_avoid_ip_fragmentation** ()

  QoS Snippet that configures RTI Connext and the UDP built-in transports (UDPv4, UDPv6, UDPv4_WAN) to avoid IP fragmentation.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_transport_udp_wan** ()

  QoS Snippet that enables the RTI Real-Time WAN Transport (UDPv4_WAN).
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_compatibility_connext_micro_version_2_4_3** ()

  QoS Snippet that configures sets LIVELINESS QoS to be compatible with RTI Connext Micro 2.4.3.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_compatibility_other_dds_vendors_enable** ()

  QoS Snippet that configures RTI Connext to interoperate with other DDS vendors.
- std::string **rti::core::builtin_profiles::qos_snippet_lib::snippet_compatibility_5_1_0_transport_enable** ()

  QoS Snippet that configures RTI Connext to interoperate with RTI Connext 5.1.0 and below for UDPv6 and SHMEM transports.

6.137.1 Detailed Description

<<***extension***>> *(*p. *354)* QoS libraries and profiles that are automatically built into RTI Connext.

The built-in profiles can be accessed in QoS XML configuration files and by using any of the APIs that accept library and profiles names by using the constants or string versions as documented on this page.

The built-in profiles are provided as a way to quickly and easily configure RTI Connext applications with a set of QoS values aimed at achieving a specific behavior.

There are three built-in QoS libraries:

- BuiltinQosLib: A library containing built-in QoS Profiles.

- BuiltinQosLibExp: A library containing experimental QoS Profiles. Experimental QoS Profiles are new QoS Profiles that have been tested internally but have not gone through an extensive validation period. Therefore, some of the settings may change in future releases based on customer and internal feedback. After validation, experimental QoS Profiles will be moved into the non-experimental library.

- BuiltinQosSnippetLib: A library containing QoS Snippets that are ready to use as elements for the QoS Profile composition pattern. For further information about this pattern visit the following article: `https://community.rti.com/best-practices/qos-profile-inheritance-and-composition-guidance`

There are three types of profiles:

- Baseline.X.X.X: These profiles represent the QoS defaults for /ndds version X.X.X. To access the defaults for the latest RTI Connext version, use the BuiltinQosLib::Baseline profile.
- Generic.X: These profiles allow you to easily configure different features and communication use-cases with RTI Connext. For example, there is a Generic.StrictReliable profile for use when your application has a requirement for no data loss.
- Pattern.X: These profiles inherit from the Generic.X profiles and allow you to configure various domain-specific communication use cases. For example, there is a Pattern.Alarm profile that can be used to manage the generation and consumption of alarm events.

There are several types of QoS Snippets. These are the current QoS Snippets available:

- Optimization.X: these QoS Snippets optimize one or more parameters related to the X QoS Policy or a specific use-case.
- QosPolicy.X.Y: these QoS Snippets set a specific QoS Policy X to the value Y.
- Feature.X: these QoS Snippets set all the needed QoS values to enable/modify a specific feature.
- Transport.X: these QoS Snippets set a specific transport defined by X. This transport may have specific scenarios that are also specified in the name.
- Compatibility.X: these QoS Snippets change the specific QoS policies to ensure compatibility with specific products or versions specified by X.

These QoS Profiles can be used as base profiles in XML configuration, then QoS Snippets can modify specific aspects of this base QoS Profile, and finally files and values can be modified to fit a specific system's needs. For example:

```
<qos_profile name="MyProfile" base_name="BuiltinQosLib::Generic.Common">

   <domain_participant_qos base_name="BuiltinQosLib::Generic.Monitoring.Common">

     <base_name>

       <element>BuiltinQosSnippetLib::Feature.Monitoring.Enable</element>

     </base_name>

     <!-- Override and add values -->

   </domain_participant_qos>

</qos_profile>
```

These names can also be used to look up the built-in QoS Profiles and QoS Snippets in the default QosProvider (**dds::core::QosProvider::Default()** (p. 2036)), for example:

```
dds::sub::qos::DataReaderQos reader_qos =
```

```
    dds::core::QosProvider::Default().datareader_qos(
        rti::core::builtin_profiles::qos_lib::strict_reliable());
dds::sub::qos::DataReaderQos reader_qos =
    dds::core::QosProvider::Default().datareader_qos(
        rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_monitoring_enable());
```

Note that the QoS Profile and QoS Snippets names also contain the library name. For example `strict_reliable()` returns "`BuiltinQosLib::Generic.StrictReliable`"

The QoS Profiles and QoS Snippets are contained in three different libraries, in three namespaces:

- **rti::core::builtin_profiles::qos_lib** (p. 712),
- **rti::core::builtin_profiles::qos_lib_exp** (p. 714), and
- **rti::core::builtin_profiles::qos_snippet_lib** (p. 716)

All the built-in QoS Profiles and QoS Snippets are documented in the BaselineRoot.documentationONLY.xml and BuiltinProfiles.documentationONLY.xml files that are included in the NDDSHOME/xml directory of the RTI Connext installation.

- BaselineRoot.documentationONLY.xml contains the root baseline QoS profile that corresponds to the default values of RTI Connext 5.0.0.
- BuiltinProfiles.documentationONLY.xml contains the rest of the built-in Qos Profiles and QoS Snippets.

6.137.2 Function Documentation

6.137.2.1 library_name() [1/3]

```
std::string rti::core::builtin_profiles::qos_lib::library_name ( )
```

A library of non-experimental QoS profiles.

String-version: "BuiltinQosLib"

6.137.2.2 baseline()

```
std::string rti::core::builtin_profiles::qos_lib::baseline ( )
```

The most up-to-date QoS default values.

You can use this profile if you want your application to pick up and use any new QoS default settings each time a new RTI Connext version is released – without changing your application code.

String-version: "Baseline"

6.137.2.3 baseline_5_0_0()

```
std::string rti::core::builtin_profiles::qos_lib::baseline_5_0_0 ( )
```

The QoS default values for version 5.0.0.

String-version: "Baseline.5.0.0"

6.137.2.4 baseline_5_1_0()

```
std::string rti::core::builtin_profiles::qos_lib::baseline_5_1_0 ( )
```

The QoS default values for version 5.1.0.

String-version: "Baseline.5.1.0"

6.137.2.5 baseline_5_2_0()

```
std::string rti::core::builtin_profiles::qos_lib::baseline_5_2_0 ( )
```

The QoS default values for version 5.2.0.

String-version: "Baseline.5.2.0"

6.137.2.6 baseline_5_3_0()

```
std::string rti::core::builtin_profiles::qos_lib::baseline_5_3_0 ( )
```

The QoS default values for version 5.3.0.

String-version: "Baseline.5.3.0"

6.137.2.7 baseline_6_0_0()

```
std::string rti::core::builtin_profiles::qos_lib::baseline_6_0_0 ( )
```

The QoS default values for version 6.0.0.

String-version: "Baseline.6.0.0"

6.137.2.8 baseline_6_1_0()

```
std::string rti::core::builtin_profiles::qos_lib::baseline_6_1_0 ( )
```

The QoS default values for version 6.1.0.

String-version: "Baseline.6.1.0"

6.137.2.9 baseline_7_0_0()

```
std::string rti::core::builtin_profiles::qos_lib::baseline_7_0_0 ( )
```

The QoS default values for version 7.0.0.

String-version: "Baseline.7.0.0"

6.137.2.10 baseline_7_1_0()

```
std::string rti::core::builtin_profiles::qos_lib::baseline_7_1_0 ( )
```

The QoS default values for version 7.1.0.

String-version: "Baseline.7.1.0"

6.137.2.11 generic_common()

```
std::string rti::core::builtin_profiles::qos_lib::generic_common ( )
```

A common Participant base profile.

All Generic.X and Pattern.X profiles inherit from this profile.

String-version: "Generic.Common"

6.137.2.12 generic_monitoring_common()

```
std::string rti::core::builtin_profiles::qos_lib::generic_monitoring_common ( )
```

Enables RTI Monitoring Library.

Generic Base participant QoS Profile that enables RTI Monitoring Library.

Use of this QoS Profile is deprecated. It is included for backwards compatibility.

Instead of using the (deprecated) "BuiltinQosLib::Generic.Monitoring.Common", apply the QoS Snippet "BuiltinQos↩ SnippetLib::Feature.Monitoring.Enable" via composition. For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosSnippetLib::Feature.Monitoring.Enable</element>

    </base_name>

</qos_profile>
```

Legacy applications may use this QoS Profile via inheritance. To do this, create a derived QoS Profile that inherits from this one. For example:

```
<qos_profile name="MyProfile" base_name="BuiltinQosLib::Generic.StrictReliable">

    <domain_participant_qos base_name="BuiltinQosLib::Generic.Monitoring.Common">

</qos_profile>
```

String-version: "Generic.Monitoring.Common"

6.137.2.13 generic_connext_micro_compatibility()

```
std::string rti::core::builtin_profiles::qos_lib::generic_connext_micro_compatibility ( )
```

Sets the values necessary to communicate with RTI Connext Micro.

This profile will always represent the QoS values required for interoperability between the most recent version of RTI Connext Micro at the time of release of the most recent version of RTI Connext.

String-version: "Generic.ConnextMicroCompatibility"

6.137.2.14 generic_connext_micro_compatibility_2_4_9()

```
std::string rti::core::builtin_profiles::qos_lib::generic_connext_micro_compatibility_2_4_9 ( )
```

Sets the values necessary to communicate with RTI Connext Micro versions 2.4.4 through at least 2.4.9.

At the time of the release of RTI Connext 5.3.0 it was not necessary to set any QoS values in order to interoperate with RTI Connext Micro. This applies to RTI Connext Micro versions 2.4.4 and later. The most recent version of RTI Connext Micro at the time of release of RTI Connext 5.3.0 was 2.4.9. There is no guarantee that this profile will interoperate with versions of RTI Connext Micro after 2.4.9.

String-version: "Generic.ConnextMicroCompatibility.2.4.9"

6.137.2.15 generic_connext_micro_compatibility_2_4_3()

```
std::string rti::core::builtin_profiles::qos_lib::generic_connext_micro_compatibility_2_4_3 ( )
```

Sets the values necessary to communicate with RTI Connext Micro versions 2.4.3 and ealier.

RTI Connext Micro versions 2.4.3 and earlier only supported the **dds::core::policy::LivelinessKind_def::MANUAL↩_BY_TOPIC** (p. 1655) LivelinessQos kind. In order to be compatible with these versions of RTI Connext Micro, the **dds::sub::DataReader** (p. 976) and **dds::pub::DataWriter** (p. 1138) must have their Liveliness kind changed to this value because the default kind in RTI Connext is **dds::core::policy::LivelinessKind_def::AUTOMATIC** (p. 1655).

String-version: "Generic.ConnextMicroCompatibility.2.4.3"

6.137.2.16 generic_other_dds_vendor_compatibility()

```
std::string rti::core::builtin_profiles::qos_lib::generic_other_dds_vendor_compatibility ( )
```

Sets the values necessary to interoperate with other DDS vendors.

String-version: "Generic.OtherDDSVendorCompatibility"

6.137.2.17 generic_510_transport_compatibility()

```
std::string rti::core::builtin_profiles::qos_lib::generic_510_transport_compatibility ( )
```

Sets the values necessary to interoperate with RTI Connext 5.1.0 using the UDPv6 and/or SHMEM transports.

String-version: "Generic.510TransportCompatibility"

6.137.2.18 generic_security()

```
std::string rti::core::builtin_profiles::qos_lib::generic_security ( )
```

Loads the DDS Secure builtin plugins.

Generic Base Participant Profile that enables the builtin DDS Security Plugins.

Use of this QoS Profile is deprecated. It is included for backwards compatibility.

Instead of using the (deprecated) "BuiltinQosLib::Generic.Security", apply the QoS Snippet "BuiltinQosSnippetLib::↩ Feature.Security.Enable" via composition. For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosSnippetLib::Feature.Security.Enable</element>

    </base_name>

</qos_profile>
```

Legacy applications may use this QoS Profile via inheritance. To do this, create a derived QoS Profile that inherits from this one. For example:

```
<qos_profile name="MyProfile" base_name="BuiltinQosLib::Generic.StrictReliable">

    <domain_participant_qos base_name="BuiltinQosLib::Generic.Security">

</qos_profile>
```

String-version: "Generic.Security"

6.137.2.19 generic_strict_reliable() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_strict_reliable ( )
```

Enables strict reliability.

Configures communication to be "strict reliable" where every sample is reliably delivered.

Combines the use of the **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) for **dds::core::policy::↩ Reliability** (p. 2149) with a **dds::core::policy::HistoryKind_def::KEEP_ALL** (p. 1607) for the **dds::core::policy::↩ HistoryKind_def** (p. 1606).

This QoS Profile also optimizes the reliability protocol setting for this configuration.

String-version: "Generic.StrictReliable"

6.137.2.20 generic_keep_last_reliable() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable ( )
```

Enables keep-last reliability.

Like the Generic.StrictReliable profile, this profile ensures in-order delivery of samples. However, new data can overwrite data that has not yet been acknowledged by the reader, therefore causing possible sample loss.

String-version: "Generic.KeepLastReliable"

6.137.2.21 generic_best_effort() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_best_effort ( )
```

Enables best-effort reliability kind.

This profile enables best-effort communication. No effort or resources are spent to track whether or not sent samples are received. Minimal resources are used. This is the most deterministic method of sending data since there is no indeterministic delay that can be introduced by resending data. Data samples may be lost. This setting is good for periodic data.

String-version: "Generic.BestEffort"

6.137.2.22 generic_strict_reliable_high_throughput() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_strict_reliable_high_throughput ( )
```

A profile that can be used to achieve high throughput.

This QoS Profile extends the **rti::core::builtin_profiles::qos_lib::generic_strict_reliable()** (p. 473) QoS Profile to perform additional, fine-grained performance tuning specific to applications that send continuously streaming data. The parameters specified here add to and/or override the parameters specified in the **rti::core::builtin_profiles::qos_lib↩ ::generic_strict_reliable()** (p. 473) QoS Profile..

String-version: "Generic.StrictReliable.HighThroughput"

6.137.2.23 generic_strict_reliable_low_latency() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_strict_reliable_low_latency ( )
```

A profile that can be used to achieve low latency.

This QoS Profile extends the **rti::core::builtin_profiles::qos_lib::generic_strict_reliable()** (p. 473) QoS Profile to perform additional, fine-grained performance tuning specific to applications that send continuously streaming data. The parameters specified here add to and/or override the parameters specified in the **rti::core::builtin_profiles::qos_lib↩ ::generic_strict_reliable()** (p. 473) QoS Profile.

String-version: "Generic.StrictReliable.LowLatency"

6.137.2.24 generic_participant_large_data() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_participant_large_data ( )
```

A common Participant base profile to facilitate sending large data.

This is a common Participant base QoS Profile that configures 3 different flow controllers: 838, 209, and 52 Mbps that can each be used to throttle application data flow at different rates.

String-version: "Generic.Participant.LargeData"

6.137.2.25 generic_participant_large_data_monitoring() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_participant_large_data_monitoring ( )
```

Configures Participants for large data *and* monitoring.

This is a common base Participant QoS Profile to configure Participants to both handle large data and use RTI Monitoring Library.

This QoS Profile is deprecated. It is included for backwards compatibility.

It is recommended that instead of inheriting from this QoS Profile, new applications apply the following QoS Snippets to their application-specific QoS Profiles:

Enable Monitoring

- **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_monitoring_enable()** (p. 501)

Enable Large Data optimizations

- **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_large_data()** (p. 493)
- **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_dynamicmemalloc()** (p. 494)

String-version: "Generic.Participant.LargeData.Monitoring"

See also

rti::core::builtin_profiles::qos_lib::generic_participant_large_data() (p. 474)

rti::core::builtin_profiles::qos_lib::generic_monitoring_common() (p. 471)

6.137.2.26 generic_strict_reliable_large_data() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_strict_reliable_large_data ( )
```

Configures endpoints for sending large data with strict reliability.

This QoS Profile extends the **rti::core::builtin_profiles::qos_lib::generic_participant_large_data()** (p. 474) QoS Profile to handle sending large samples reliably. This QoS Profile optimizes memory usage per sample within RTI Connext. You can use this QoS Profile directly, which doesn't perform any flow control, or you can use one of the three QoS Profiles below (Generic.StrictReliable.LargeData.∗Flow), which use this QoS Profile as a common base QoS Profile, but additionally enable asynchronous publication. Each of these three QoS Profiles uses one of the three flow controllers defined in **rti::core::builtin_profiles::qos_lib::generic_participant_large_data()** (p. 474) in order to throttle application data flow.

String-version: "Generic.StrictReliable.LargeData"

6.137.2.27 generic_keep_last_reliable_large_data() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_large_data ( )
```

Configures endpoints for sending large data with keep-last reliability.

This QoS Profile is similar to the **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable()** (p. 473) QoS Profile, but also adds QoS Snippets to handle sending large samples reliably. You can use this QoS Profile directly, which doesn't perform any flow control, or you can use one of the three QoS Profiles below (Generic.KeepLastReliable.↩ LargeData.∗Flow), which use this QoS Profile as a common base QoS Profile, but additionally enable asynchronous publication. Each of these three QoS Profiles uses one of the three flow controllers defined in **rti::core::builtin_↩ profiles::qos_lib::generic_participant_large_data()** (p. 474) in order to throttle application data flow.

String-version: "Generic.KeepLastReliable.LargeData"

6.137.2.28 generic_strict_reliable_large_data_fast_flow() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_strict_reliable_large_data_fast_flow ( )
```

Configures strictly reliable communication for large data with a fast flow controller.

Strictly reliable communication for large data with a 838 Mbps ($\sim$ 100 MB/sec) flow controller.

String-version: "Generic.StrictReliable.LargeData.FastFlow"

6.137.2.29 generic_strict_reliable_large_data_medium_flow() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_strict_reliable_large_data_medium_flow (
)
```

Configures strictly reliable communication for large data with a medium flow controller.

Strictly reliable communication for large data with a 209Mbps ($\sim$ 25 MB/sec) flow controller.

String-version: "Generic.StrictReliable.LargeData.MediumFlow"

6.137.2.30 generic_strict_reliable_large_data_slow_flow() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_strict_reliable_large_data_slow_flow ( )
```

Configures strictly reliable communication for large data with a slow flow controller.

Strictly reliable communication for large data with a 52 MB/sec (∼ 6.25 MB/sec) flow controller.

String-version: "Generic.StrictReliable.LargeData.SlowFlow"

6.137.2.31 generic_keep_last_reliable_large_data_fast_flow() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_large_data_fast_flow
( )
```

Configures keep-last reliable communication for large data with a fast flow controller.

Keep-last reliable communication for large data with a 838 Mbps (∼ 100 MB/sec) flow controller.

String-version: "Generic.KeepLastReliable.LargeData.FastFlow"

6.137.2.32 generic_keep_last_reliable_large_data_medium_flow() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_large_data_medium_↩
flow ( )
```

Configures keep-last reliable communication for large data with a medium flow controller.

Keep-last reliable communication for large data with a 209Mbps (∼ 25 MB/sec) flow controller.

String-version: "Generic.KeepLastReliable.LargeData.MediumFlow"

6.137.2.33 generic_keep_last_reliable_large_data_slow_flow() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_large_data_slow_flow
( )
```

Configures keep-last reliable communication for large data with a slow flow controller.

Keep-last reliable communication for large data with a 52 MB/sec (∼ 6.25 MB/sec) flow controller.

String-version: "Generic.KeepLastReliable.LargeData.SlowFlow"

6.137.2.34 generic_keep_last_reliable_transient_local() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_transient_local ( )
```

Persists the samples of a DataWriter as long as the entity exists.

This profile extends the **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable()** (p. 473) profile, but persists the samples of a **dds::pub::DataWriter** (p. 1138) as long as the entity exists in order to deliver them to late-joining DataReaders.

String-version: "Generic.KeepLastReliable.TransientLocal"

6.137.2.35 generic_keep_last_reliable_transient() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_transient ( )
```

Persists samples using RTI Persistence Service.

This profile extends the **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable()** (p. 473) profile, but persists samples using Persistence Service in order to deliver them to late-joining DataReaders.

String-version: "Generic.KeepLastReliable.Transient"

6.137.2.36 generic_keep_last_reliable_persistent() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_persistent ( )
```

Persists samples in permanent storage, like a disk, using RTI Persistence Service.

This profile extends the **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable()** (p. 473) profile, but persists samples in permanent storage, such as a disk, using Persistence Service in order to deliver them to late-joining Data↩ Readers.

String-version: "Generic.KeepLastReliable.Persistent"

6.137.2.37 generic_auto_tuning() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_auto_tuning ( )
```

Enables the Turbo Mode batching and Auto Throttle experimental features.

Turbo Mode batching adjusts the maximum number of bytes of a batch based on how frequently samples are being written. Auto Throttle auto-adjusts the speed at which a writer will write samples, based on the number of unacknowledged samples in its queue.

These features are designed to auto-adjust the publishing behavior within a system in order to achieve the best possible performance with regards to throughput and latency.

String-version: "Generic.AutoTuning"

6.137.2.38 generic_minimal_memory_footprint() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::generic_minimal_memory_footprint ( )
```

Uses a set of QoS which reduces the memory footprint of the application.

Uses a set of QoS which reduces the memory footprint of the application.

String-version: "Generic.MinimalMemoryFootprint"

6.137.2.39 generic_monitoring2()

```
std::string rti::core::builtin_profiles::qos_lib::generic_monitoring2 ( )
```

The default QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use.

This profile inherits from **rti::core::builtin_profiles::qos_lib::generic_minimal_memory_footprint()** (p. 478).

The following DomainParticipant QoS policy of this profile cannot be modified and it is overwritten by the RTI Monitoring Library 2.0:

- Discovery Config Built-in Channel Kind.

The following DataWriter and DataReader QoS policies of this profile cannot be modified and they are overwritten by the RTI Monitoring Library 2.0:

- Reliability Kind.
- Durability Kind.
- History Kind.
- Publish Mode Kind.
- Protocol -> RTPS Reliable Writer -> Max Heartbeat Retries.

This profile uses Topic Filters to select the DataWriter and DataReader QoS depending on the Observability Distribution Topic.

You should be using this profile or a profile inheriting from it when you configure the distribution of telemetry data using the <distribution_settings> tag under <monitoring> for the <participant_factory_qos>.

Note: This profile does not enable the use of the RTI Monitoring Library 2.0. To do that you can use the Snippet **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_monitoring2_enable()** (p. 501).

String-version: "Generic.Monitoring2"

See also

MONITORING_PERIODIC_TOPIC_NAME
MONITORING_EVENT_TOPIC_NAME
MONITORING_LOGGING_TOPIC_NAME

6.137.2.40 generic_monitoring2_wan()

```
std::string rti::core::builtin_profiles::qos_lib::generic_monitoring2_wan ( )
```

The QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use to communicate over WAN with a Collector Service.

This profile inherits from **rti::core::builtin_profiles::qos_lib::generic_monitoring2()** (p. 479).

Note: This profile does not enable the use of the RTI Monitoring Library 2.0. To do that you can use the Snippet **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_monitoring2_enable()** (p. 501).

String-version: "Generic.Monitoring2.WAN"

See also

rti::core::builtin_profiles::qos_lib::generic_monitoring2() (p. 479)

6.137.2.41 pattern_rpc()

```
std::string rti::core::builtin_profiles::qos_lib::pattern_rpc ( )
```

This profile contains the default RPC and Request-Reply QoS settings.

This profile contains the QoS that is used to create the DataReaders and DataWriters in a Requester, a Replier, a Client or a Service.

Use this profile as the base profile if you need to customize the QoS settings of any of these entities.

String-version: "Pattern.RPC"

6.137.2.42 pattern_periodic_data() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::pattern_periodic_data ( )
```

Used for applications that expect periodic data.

This QoS Profile is intended to be used for applications that expect periodic data such as sensor data. The deadline that is set in this profile can be used to detect when DataWriters are not publishing data with the expected periodicity.

String-version: "Pattern.PeriodicData"

6.137.2.43 pattern_streaming() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::pattern_streaming ( )
```

Used for applications that stream data.

The data sent in streaming applications is commonly periodic. Therefore this profile simply inherits from the **rti::core::builtin_profiles::qos_lib::pattern_periodic_data()** (p. 480) profile. Note: With this QoS Profile, the application may lose data, which may be acceptable in use cases such as video conferencing.

String-version: "Pattern.Streaming"

6.137.2.44 pattern_reliable_streaming() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::pattern_reliable_streaming ( )
```

Used for applications that stream data *and* require reliable communication.

Sometimes streaming applications require reliable communication while still tolerating some data loss. In this case, we inherit from the **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable()** (p. 473) QoS Profile and the following QoS Snippets:

- **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_reliability_reliable()** (p. 496)
- **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_history_keep_last_1()** (p. 497)
- **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_keep_last()** (p. 492)

This QoS Snippet also increases the **dds::core::policy::History::depth** (p. 1605) to reduce the probability of losing samples.

String-version: "Pattern.ReliableStreaming"

6.137.2.45 pattern_event() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::pattern_event ( )
```

Used for applications that handle events.

This QoS Profile can be used by applications in which samples represent events such as button pushes or alerts. When events are triggered, the system should almost always do something, meaning that you don't want the system to lose the event. This means that the system requires strictly reliable communication. To enable it, use the following QoS Snippets:

- **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_reliability_reliable()** (p. 496)
- **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_history_keep_all()** (p. 497)
- **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_keep_all()** (p. 491)

Since events and alerts are critical and non-periodic data, it is important to detect situations in which communication between a **dds::pub::DataWriter** (p. 1138) and **dds::sub::DataReader** (p. 976) is broken. This is why this profile sets the **dds::core::policy::Liveliness** (p. 1645). If the **dds::pub::DataWriter** (p. 1138) does not assert its liveliness in a timely manner, the **dds::sub::DataReader** (p. 976) will report 'loss of liveliness' to the application.

String-version: "Pattern.Event"

6.137.2.46 pattern_alarm_event() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::pattern_alarm_event ( )
```

Used for applications that handle alarm events.

An alarm is a type of event; therefore this profile simply inherits from **rti::core::builtin_profiles::qos_lib::pattern_↩event()** (p. 481).

String-version: "Pattern.AlarmEvent"

6.137.2.47 pattern_status() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::pattern_status ( )
```

Used for applications whose samples represent statuses.

This QoS Profile can be used by applications in which samples represent state variables whose values remain valid as long as they don't explicitly change. State variables typically do not change periodically. State variables and their values should also be available to applications that appear after the value originally changed because it is unreasonable to have to wait until the next change of state, which may be indeterminate.

Whether to use this QoS Profile or **rti::core::builtin_profiles::qos_lib::pattern_periodic_data()** (p. 480) can often be an application choice. For example, if a DataWriter is publishing temperature sensor data, it could use the **rti::core↩::builtin_profiles::qos_lib::pattern_periodic_data()** (p. 480) QoS Profile and publish the data at a fixed rate or it could use the **rti::core::builtin_profiles::qos_lib::pattern_status()** (p. 482) QoS Profile and only publish the temperature when it changes more than 1 degree.

String-version: "Pattern.Status"

6.137.2.48 pattern_alarm_status() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::pattern_alarm_status ( )
```

Used for applications in which samples represent alarm statuses.

An alarm status is a type of status; therefore this QoS Profile simply inherits from **rti::core::builtin_profiles::qos_lib↩::pattern_status()** (p. 482).

String-version: "Pattern.AlarmStatus"

6.137.2.49 pattern_last_value_cache() [1/2]

```
std::string rti::core::builtin_profiles::qos_lib::pattern_last_value_cache ( )
```

Used for applications that only need the last published value.

With this QoS Profile, a **dds::pub::DataWriter** (p. 1138) will keep in its queue the last value that was published for each sample instance. Late-joining DataReaders will get that value when they join the system. This QoS Profile inherits from **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_transient_local()** (p. 477) because the use case requires delivery to late-joiners.

String-version: "Pattern.LastValueCache"

6.137.2.50 library_name() [2/3]

```
std::string rti::core::builtin_profiles::qos_lib_exp::library_name ( )
```

A library of experimental QoS profiles.

Experimental profiles are new profiles that have been tested internally but have not gone through an extensive validation period. Therefore some of the settings may change in future releases based on customer and internal feedback. After validation, experimental profiles will be moved into the non-experimental library.

QoS Profiles in this library are deprecated. They have been moved to "BuiltinQosLib". You should use the QoS Profiles from "BuiltinQosLib" instead of the ones in "BuiltinQosLibExp". The experimental profiles are still defined here to avoid backward compatibility issues.

String-version: "BuiltinQosLibExp"

6.137.2.51 generic_strict_reliable() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable ( )
```

Enables strict reliability.

Configures communication to be "strict reliable" where every sample is reliably delivered.

Combines the use of the **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) for **dds::core::policy::↩ Reliability** (p. 2149) with a **dds::core::policy::HistoryKind_def::KEEP_ALL** (p. 1607) for the **dds::core::policy::↩ HistoryKind_def** (p. 1606).

This QoS Profile also optimizes the reliability protocol setting for this configuration.

String-version: "Generic.StrictReliable"

6.137.2.52 generic_keep_last_reliable() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable ( )
```

Enables keep-last reliability.

Like the Generic.StrictReliable profile, this profile ensures in-order delivery of samples. However, new data can overwrite data that has not yet been acknowledged by the reader, therefore causing possible sample loss.

String-version: "Generic.KeepLastReliable"

6.137.2.53 generic_best_effort() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_best_effort ( )
```

Enables best-effort reliability kind.

This profile enables best-effort communication. No effort or resources are spent to track whether or not sent samples are received. Minimal resources are used. This is the most deterministic method of sending data since there is no indeterministic delay that can be introduced by resending data. Data samples may be lost. This setting is good for periodic data.

String-version: "Generic.BestEffort"

6.137.2.54 generic_strict_reliable_high_throughput() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable_high_throughput ( )
```

A profile that can be used to achieve high throughput.

This QoS Profile extends the **rti::core::builtin_profiles::qos_lib::generic_strict_reliable()** (p. 473) QoS Profile to perform additional, fine-grained performance tuning specific to applications that send continuously streaming data. The parameters specified here add to and/or override the parameters specified in the **rti::core::builtin_profiles::qos_lib↩ ::generic_strict_reliable()** (p. 473) QoS Profile..

String-version: "Generic.StrictReliable.HighThroughput"

6.137.2.55 generic_strict_reliable_low_latency() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable_low_latency ( )
```

A profile that can be used to achieve low latency.

This QoS Profile extends the **rti::core::builtin_profiles::qos_lib::generic_strict_reliable()** (p. 473) QoS Profile to perform additional, fine-grained performance tuning specific to applications that send continuously streaming data. The parameters specified here add to and/or override the parameters specified in the **rti::core::builtin_profiles::qos_lib↩ ::generic_strict_reliable()** (p. 473) QoS Profile.

String-version: "Generic.StrictReliable.LowLatency"

6.137.2.56 generic_participant_large_data() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_participant_large_data ( )
```

A common Participant base profile to facilitate sending large data.

This is a common Participant base QoS Profile that configures 3 different flow controllers: 838, 209, and 52 Mbps that can each be used to throttle application data flow at different rates.

String-version: "Generic.Participant.LargeData"

6.137.2.57 generic_participant_large_data_monitoring() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_participant_large_data_monitoring (
)
```

Configures Participants for large data *and* monitoring.

This is a common base Participant QoS Profile to configure Participants to both handle large data and use RTI Monitoring Library.

This QoS Profile is deprecated. It is included for backwards compatibility.

It is recommended that instead of inheriting from this QoS Profile, new applications apply the following QoS Snippets to their application-specific QoS Profiles:

Enable Monitoring

- **rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_monitoring_enable()** (p. 501)

Enable Large Data optimizations

- **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_large_data()** (p. 493)
- **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_dynamicmemalloc()** (p. 494)

String-version: "Generic.Participant.LargeData.Monitoring"

See also

rti::core::builtin_profiles::qos_lib::generic_participant_large_data() (p. 474)

rti::core::builtin_profiles::qos_lib::generic_monitoring_common() (p. 471)

6.137.2.58 generic_strict_reliable_large_data() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable_large_data ( )
```

Configures endpoints for sending large data with strict reliability.

This QoS Profile extends the **rti::core::builtin_profiles::qos_lib::generic_participant_large_data()** (p. 474) QoS Profile to handle sending large samples reliably. This QoS Profile optimizes memory usage per sample within RTI Connext. You can use this QoS Profile directly, which doesn't perform any flow control, or you can use one of the three QoS Profiles below (Generic.StrictReliable.LargeData.∗Flow), which use this QoS Profile as a common base QoS Profile, but additionally enable asynchronous publication. Each of these three QoS Profiles uses one of the three flow controllers defined in **rti::core::builtin_profiles::qos_lib::generic_participant_large_data()** (p. 474) in order to throttle application data flow.

String-version: "Generic.StrictReliable.LargeData"

6.137.2.59 generic_keep_last_reliable_large_data() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_large_data ( )
```

Configures endpoints for sending large data with keep-last reliability.

This QoS Profile is similar to the **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable()** (p. 473) QoS Profile, but also adds QoS Snippets to handle sending large samples reliably. You can use this QoS Profile directly, which doesn't perform any flow control, or you can use one of the three QoS Profiles below (Generic.KeepLastReliable.↩ LargeData.∗Flow), which use this QoS Profile as a common base QoS Profile, but additionally enable asynchronous publication. Each of these three QoS Profiles uses one of the three flow controllers defined in **rti::core::builtin_↩ profiles::qos_lib::generic_participant_large_data()** (p. 474) in order to throttle application data flow.

String-version: "Generic.KeepLastReliable.LargeData"

6.137.2.60 generic_strict_reliable_large_data_fast_flow() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable_large_data_fast_flow
( )
```

Configures strictly reliable communication for large data with a fast flow controller.

Strictly reliable communication for large data with a 838 Mbps ($\sim$ 100 MB/sec) flow controller.

String-version: "Generic.StrictReliable.LargeData.FastFlow"

6.137.2.61 generic_strict_reliable_large_data_medium_flow() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable_large_data_medium_↩
flow ( )
```

Configures strictly reliable communication for large data with a medium flow controller.

Strictly reliable communication for large data with a 209Mbps ($\sim$ 25 MB/sec) flow controller.

String-version: "Generic.StrictReliable.LargeData.MediumFlow"

6.137.2.62 generic_strict_reliable_large_data_slow_flow() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_strict_reliable_large_data_slow_flow
( )
```

Configures strictly reliable communication for large data with a slow flow controller.

Strictly reliable communication for large data with a 52 MB/sec ($\sim$ 6.25 MB/sec) flow controller.

String-version: "Generic.StrictReliable.LargeData.SlowFlow"

6.137.2.63 generic_keep_last_reliable_large_data_fast_flow() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_large_data_fast↩
_flow ( )
```

Configures keep-last reliable communication for large data with a fast flow controller.

Keep-last reliable communication for large data with a 838 Mbps (∼ 100 MB/sec) flow controller.

String-version: "Generic.KeepLastReliable.LargeData.FastFlow"

6.137.2.64 generic_keep_last_reliable_large_data_medium_flow() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_large_data_↩
medium_flow ( )
```

Configures keep-last reliable communication for large data with a medium flow controller.

Keep-last reliable communication for large data with a 209Mbps (∼ 25 MB/sec) flow controller.

String-version: "Generic.KeepLastReliable.LargeData.MediumFlow"

6.137.2.65 generic_keep_last_reliable_large_data_slow_flow() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_large_data_slow↩
_flow ( )
```

Configures keep-last reliable communication for large data with a slow flow controller.

Keep-last reliable communication for large data with a 52 MB/sec (∼ 6.25 MB/sec) flow controller.

String-version: "Generic.KeepLastReliable.LargeData.SlowFlow"

6.137.2.66 generic_keep_last_reliable_transient_local() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_transient_local (
)
```

Persists the samples of a DataWriter as long as the entity exists.

This profile extends the **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable()** (p. 473) profile, but persists the samples of a **dds::pub::DataWriter** (p. 1138) as long as the entity exists in order to deliver them to late-joining DataReaders.

String-version: "Generic.KeepLastReliable.TransientLocal"

6.137.2.67 generic_keep_last_reliable_transient() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_transient ( )
```

Persists samples using RTI Persistence Service.

This profile extends the **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable()** (p. 473) profile, but persists samples using Persistence Service in order to deliver them to late-joining DataReaders.

String-version: "Generic.KeepLastReliable.Transient"

6.137.2.68 generic_keep_last_reliable_persistent() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_keep_last_reliable_persistent ( )
```

Persists samples in permanent storage, like a disk, using RTI Persistence Service.

This profile extends the **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable()** (p. 473) profile, but persists samples in permanent storage, such as a disk, using Persistence Service in order to deliver them to late-joining Data↩ Readers.

String-version: "Generic.KeepLastReliable.Persistent"

6.137.2.69 generic_auto_tuning() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_auto_tuning ( )
```

Enables the Turbo Mode batching and Auto Throttle experimental features.

Turbo Mode batching adjusts the maximum number of bytes of a batch based on how frequently samples are being written. Auto Throttle auto-adjusts the speed at which a writer will write samples, based on the number of unacknowledged samples in its queue.

These features are designed to auto-adjust the publishing behavior within a system in order to achieve the best possible performance with regards to throughput and latency.

String-version: "Generic.AutoTuning"

6.137.2.70 generic_minimal_memory_footprint() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::generic_minimal_memory_footprint ( )
```

Uses a set of QoS which reduces the memory footprint of the application.

Uses a set of QoS which reduces the memory footprint of the application.

String-version: "Generic.MinimalMemoryFootprint"

6.137.2.71 pattern_periodic_data() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::pattern_periodic_data ( )
```

Used for applications that expect periodic data.

This QoS Profile is intended to be used for applications that expect periodic data such as sensor data. The deadline that is set in this profile can be used to detect when DataWriters are not publishing data with the expected periodicity.

String-version: "Pattern.PeriodicData"

6.137.2.72 pattern_streaming() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::pattern_streaming ( )
```

Used for applications that stream data.

The data sent in streaming applications is commonly periodic. Therefore this profile simply inherits from the **rti::core::builtin_profiles::qos_lib::pattern_periodic_data()** (p. 480) profile. Note: With this QoS Profile, the application may lose data, which may be acceptable in use cases such as video conferencing.

String-version: "Pattern.Streaming"

6.137.2.73 pattern_reliable_streaming() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::pattern_reliable_streaming ( )
```

Used for applications that stream data *and* require reliable communication.

Sometimes streaming applications require reliable communication while still tolerating some data loss. In this case, we inherit from the **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable()** (p. 473) QoS Profile and the following QoS Snippets:

- **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_reliability_reliable()** (p. 496)
- **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_history_keep_last_1()** (p. 497)
- **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_keep_last()** (p. 492)

This QoS Snippet also increases the **dds::core::policy::History::depth** (p. 1605) to reduce the probability of losing samples.

String-version: "Pattern.ReliableStreaming"

6.137.2.74 pattern_event() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::pattern_event ( )
```

Used for applications that handle events.

This QoS Profile can be used by applications in which samples represent events such as button pushes or alerts. When events are triggered, the system should almost always do something, meaning that you don't want the system to lose the event. This means that the system requires strictly reliable communication. To enable it, use the following QoS Snippets:

- **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_reliability_reliable()** (p. 496)
- **rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_history_keep_all()** (p. 497)
- **rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_protocol_keep_all()** (p. 491)

Since events and alerts are critical and non-periodic data, it is important to detect situations in which communication between a **dds::pub::DataWriter** (p. 1138) and **dds::sub::DataReader** (p. 976) is broken. This is why this profile sets the **dds::core::policy::Liveliness** (p. 1645). If the **dds::pub::DataWriter** (p. 1138) does not assert its liveliness in a timely manner, the **dds::sub::DataReader** (p. 976) will report 'loss of liveliness' to the application.

String-version: "Pattern.Event"

6.137.2.75 pattern_alarm_event() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::pattern_alarm_event ( )
```

Used for applications that handle alarm events.

An alarm is a type of event; therefore this profile simply inherits from **rti::core::builtin_profiles::qos_lib::pattern_event()** (p. 481).

String-version: "Pattern.AlarmEvent"

6.137.2.76 pattern_status() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::pattern_status ( )
```

Used for applications whose samples represent statuses.

This QoS Profile can be used by applications in which samples represent state variables whose values remain valid as long as they don't explicitly change. State variables typically do not change periodically. State variables and their values should also be available to applications that appear after the value originally changed because it is unreasonable to have to wait until the next change of state, which may be indeterminate.

Whether to use this QoS Profile or **rti::core::builtin_profiles::qos_lib::pattern_periodic_data()** (p. 480) can often be an application choice. For example, if a DataWriter is publishing temperature sensor data, it could use the **rti::core::builtin_profiles::qos_lib::pattern_periodic_data()** (p. 480) QoS Profile and publish the data at a fixed rate or it could use the **rti::core::builtin_profiles::qos_lib::pattern_status()** (p. 482) QoS Profile and only publish the temperature when it changes more than 1 degree.

String-version: "Pattern.Status"

6.137.2.77 pattern_alarm_status() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::pattern_alarm_status ( )
```

Used for applications in which samples represent alarm statuses.

An alarm status is a type of status; therefore this QoS Profile simply inherits from **rti::core::builtin_profiles::qos_lib↩ ::pattern_status()** (p. 482).

String-version: "Pattern.AlarmStatus"

6.137.2.78 pattern_last_value_cache() [2/2]

```
std::string rti::core::builtin_profiles::qos_lib_exp::pattern_last_value_cache ( )
```

Used for applications that only need the last published value.

With this QoS Profile, a **dds::pub::DataWriter** (p. 1138) will keep in its queue the last value that was published for each sample instance. Late-joining DataReaders will get that value when they join the system. This QoS Profile inherits from **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_transient_local()** (p. 477) because the use case requires delivery to late-joiners.

String-version: "Pattern.LastValueCache"

6.137.2.79 library_name() [3/3]

```
std::string rti::core::builtin_profiles::qos_snippet_lib::library_name ( )
```

A library of QoS Snippets.

String-version: "BuiltinQosSnippetLib"

6.137.2.80 snippet_optimization_reliability_protocol_common()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_↩
protocol_common ( )
```

QoS Snippet that configures the reliability protocol with a common configuration.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet defines parameters common to a set of "alternative" QoS Snippets that configure the reliability protocol.

Modified QoS Parameters:

- **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::ResourceLimits** (p. 2203)
- **rti::core::policy::DataReaderProtocol::rtps_reliable_reader** (p. 1064)

This QoS Snippet configures the reliability protocol parameters for more aggressive (faster) heartbeats so that sample loss is detected and repaired faster.

This QoS Snippet also sets the **rti::core::policy::RtpsReliableWriterProtocol::max_heartbeat_retries** (p. 2239), which works in combination with the heartbeat rate to determine when the **dds::pub::DataWriter** (p. 1138) considers a **dds::sub::DataReader** (p. 976) non-responsive. This QoS Snippet configures the **rti::core::policy::DataWriter↩ Protocol::rtps_reliable_writer** (p. 1217) parameters so that the **dds::pub::DataWriter** (p. 1138) considers the **dds↩ ::sub::DataReader** (p. 976) to be "inactive" after 500 unresponded heartbeats.

String-version: "Optimization.ReliabilityProtocol.Common"

6.137.2.81 snippet_optimization_reliability_protocol_keep_all()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_↩
protocol_keep_all ( )
```

QoS Snippet that configures the reliability protocol for KEEP_ALL.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet optimizes the **dds::pub::DataWriter** (p. 1138) reliable protocol parameters for a reliable **dds::pub↩ ::DataWriter** (p. 1138) with **dds::core::policy::HistoryKind_def** (p. 1606) set to **dds::core::policy::HistoryKind_↩ def::KEEP_ALL** (p. 1607).

Modified QoS Parameters:

- **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::ResourceLimits** (p. 2203)
- **rti::core::policy::DataReaderProtocol::rtps_reliable_reader** (p. 1064)

Note that this QoS Snippet does not configure the **dds::core::policy::Reliability** (p. 2149) or **dds::core::policy::↩ History** (p. 1602) QoS policies. It is intended to be used in combination with other QoS Snippets that configure those policies.

String-version: "Optimization.ReliabilityProtocol.KeepAll"

6.137.2.82 snippet_optimization_reliability_protocol_keep_last()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_↩
protocol_keep_last ( )
```

QoS Snippet that configures the reliability protocol for KEEP_LAST.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet optimizes the **dds::pub::DataWriter** (p. 1138) reliable protocol parameters for a reliable **dds::pub↩ ::DataWriter** (p. 1138) with **dds::core::policy::HistoryKind_def** (p. 1606) set to **dds::core::policy::HistoryKind_↩ def::KEEP_LAST** (p. 1607).

Modified QoS Parameters:

- **dds::pub::qos::DataWriterQos** (p. 1226) => **dds::core::policy::ResourceLimits** (p. 2203)
- **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::ResourceLimits** (p. 2203)
- **rti::core::policy::DataReaderProtocol::rtps_reliable_reader** (p. 1064)

Note that this QoS Snippet does not configure the **dds::core::policy::Reliability** (p. 2149) or **dds::core::policy::↩ History** (p. 1602) QoS policies. It is intended to be used in combination with other QoS Snippets that configure those policies.

String-version: "Optimization.ReliabilityProtocol.KeepLast"

6.137.2.83 snippet_optimization_reliability_protocol_high_rate()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_↩
protocol_high_rate ( )
```

QoS Snippet that configures the reliability protocol for sending data at a high rate.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet optimizes the **dds::pub::DataWriter** (p. 1138) reliable protocol parameters for a reliable **dds::pub↩ ::DataWriter** (p. 1138) that is writing messages at high rates, especially in situations where throughput is favored over latency.

Modified QoS Parameters:

- **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::ResourceLimits** (p. 2203)
- **rti::core::policy::DataReaderProtocol::rtps_reliable_reader** (p. 1064)

This QoS Snippet sets a fast rate of heartbeats so that errors are detected and repaired more swiftly.

Note that to get the highest thoughout you may need to apply additional changes to the final QoS Profile. See the QoS Profile **rti::core::builtin_profiles::qos_lib::generic_strict_reliable_high_throughput()** (p. 474) for further information.

String-version: "Optimization.ReliabilityProtocol.HighRate"

6.137.2.84 snippet_optimization_reliability_protocol_low_latency()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_↩
protocol_low_latency ( )
```

QoS Snippet that configures the reliability protocol for sending data at low latency.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet modifies the Reliable Protocol parameters to accomplish low latency.

Modified QoS Parameters:

- **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::ResourceLimits** (p. 2203)
- **rti::core::policy::DataReaderProtocol::rtps_reliable_reader** (p. 1064)

Note that to get the lowest latency you may need to apply additional changes to the final QoS Profile. See the QoS Profile **rti::core::builtin_profiles::qos_lib::generic_strict_reliable_low_latency()** (p. 474) for further information.

String-version: "Optimization.ReliabilityProtocol.LowLatency"

6.137.2.85 snippet_optimization_reliability_protocol_large_data()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_↩
protocol_large_data ( )
```

QoS Snippet that configures the reliability protocol for large data.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

Modifies the Reliable Protocol parameters and Resource Limits to work better with Large Data.

Modified QoS Parameters:

- **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::ResourceLimits** (p. 2203)
- **rti::core::policy::DataReaderProtocol::rtps_reliable_reader** (p. 1064)

Note that, to send large data, it is recommended to apply additional changes that configure the data caches, transport buffers, etc. See, for example, **rti::core::builtin_profiles::qos_lib::generic_keep_last_reliable_large_data()** (p. 476), and derivatives for fully functional Large Data QoS Profiles.

String-version: "Optimization.ReliabilityProtocol.LargeData"

6.137.2.86 snippet_optimization_reliability_protocol_dynamicmemalloc()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_reliability_↩
protocol_dynamicmemalloc ( )
```

Configures the DataWriter and DataReader caches to use dynamic memory allocation for Large Data samples.

Modified QoS Parameters:

- **dds::pub::qos::DataWriterQos** (p. 1226) => **dds::core::policy::ResourceLimits** (p. 2203)
- **dds::pub::qos::DataWriterQos** (p. 1226) => **rti::core::policy::Property** (p. 1969) named "dds.data_writer.↩ history.memory_manager.∗"
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::ResourceLimits** (p. 2203)
- **rti::core::policy::DataReaderResourceLimits** (p. 1081)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **rti::core::policy::Property** (p. 1969) named "dds.data_reader.↩ history.memory_manager.∗"

This configuration is needed to handle data that contains unbounded sequences or strings. This QoS Snippet is also recommended if samples can have very different sizes and the bigger samples can be very large.

If dynamic memory allocation is not used for the larger samples, then all samples are allocated to their maximum size which can consume a lot of resources.

String-version: "Optimization.DataCache.LargeData.DynamicMemAlloc"

6.137.2.87 snippet_optimization_discovery_common()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_discovery_common (
)
```

QoS Snippet that optimizes discovery with a common configuration.

Optimizes the **dds::domain::qos::DomainParticipantQos** (p. 1384) to detect faster discovery changes. This QoS Snippet increases the speed moderately so that it fits the normal scenarios.

Modified QoS Parameters:

- **rti::core::policy::DiscoveryConfig::participant_liveliness_lease_duration** (p. 1279)
- **rti::core::policy::DiscoveryConfig::participant_liveliness_assert_period** (p. 1279)
- **rti::core::policy::DiscoveryConfig::max_liveliness_loss_detection_period** (p. 1281)
- **rti::core::policy::DiscoveryConfig::initial_participant_announcements** (p. 1282)
- **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289)
- **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291)

String-version: "Optimization.Discovery.Common"

6.137.2.88 snippet_optimization_discovery_participant_compact()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_discovery_participant↩
_compact ( )
```

QoS Snippet that optimizes the Participant QoS to send less discovery information.

Modified QoS Parameters:

- **dds::domain::qos::DomainParticipantQos** (p. 1384) => **rti::core::policy::Property** (p. 1969) named "dds.↩ participant.inter_participant.∗"
- **dds::domain::qos::DomainParticipantQos** (p. 1384) => **rti::core::policy::Property** (p. 1969) named "dds.↩ sys_info.∗"

String-version: "Optimization.Discovery.Participant.Compact"

6.137.2.89 snippet_optimization_discovery_endpoint_fast()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_discovery_endpoint↩
_fast ( )
```

QoS Snippet that optimizes the Endpoint Discovery to be faster.

This is useful when using security, to prevent a noticeable delay.

Modified QoS Parameters:

- **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289)
- **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291)

String-version: "Optimization.Discovery.Endpoint.Fast"

6.137.2.90 snippet_optimization_transport_large_buffers()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_optimization_transport_large_↩
buffers ( )
```

QoS Snippet that increases the Participant default buffer that shmem and udpv4 use.

This is useful when using Large Data

Modified QoS Parameters:

- **rti::core::policy::ReceiverPool** (p. 2144)
- **rti::core::policy::TransportBuiltin** (p. 2533)

String-version: "Optimization.Transport.LargeBuffers"

6.137.2.91 snippet_qos_policy_reliability_reliable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_reliability_reliable
( )
```

QoS Snippet that sets RELIABILITY QoS to RELIABLE.

This also configures a blocking time in case the **dds::pub::DataWriter** (p. 1138) writes faster than the DataReaders can accommodate.

Modified QoS Parameters:

- **dds::pub::qos::DataWriterQos** (p. 1226) => **dds::core::policy::Reliability** (p. 2149)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::Reliability** (p. 2149)

Note that by itself enabling reliability does not ensure that every sample written is delivered to the DataReaders. This is because the **dds::pub::DataWriter** (p. 1138) and/or **dds::sub::DataReader** (p. 976) can be configured to override samples in its cache based on the configuration of the **dds::core::policy::History** (p. 1602) QoS policy.

To ensure delivery of every sample (at the expense of potentially blocking the **dds::pub::DataWriter** (p. 1138)), use the QoS Profile **rti::core::builtin_profiles::qos_lib::generic_strict_reliable()** (p. 473) or one of the derived QoS Profiles.

String-version: "QosPolicy.Reliability.Reliable"

6.137.2.92 snippet_qos_policy_reliability_best_effort()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_reliability_best_↩
effort ( )
```

QoS Snippet that sets RELIABILITY QoS to BEST_EFFORT.

Modified QoS Parameters:

- **dds::pub::qos::DataWriterQos** (p. 1226) => **dds::core::policy::Reliability** (p. 2149)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::Reliability** (p. 2149)

With best-effort, there are no resources spent to confirm delivery of samples nor repairs of any samples that may be lost.

Best-effort communication reduces jitter; therefore, the delay between sending data and receiving it is more deterministic for the samples that are actually received. Best-effort is good for periodic data where it may be better to get the next value than to wait for the previous one to be repaired.

String-version: "QosPolicy.Reliability.BestEffort"

6.137.2.93 snippet_qos_policy_history_keep_last_1()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_history_keep_last_1 (
)
```

QoS Snippet that sets HISTORY QoS to KEEP_LAST kind with depth 1.

Modified QoS Parameters:

- **dds::pub::qos::DataWriterQos** (p. 1226) => **dds::core::policy::History** (p. 1602)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::History** (p. 1602)

String-version: "QosPolicy.History.KeepLast_1"

6.137.2.94 snippet_qos_policy_history_keep_all()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_history_keep_all ( )
```

QoS Snippet that sets HISTORY QosPolicy to KEEP_ALL kind.

Modified QoS Parameters:

- **dds::pub::qos::DataWriterQos** (p. 1226) => **dds::core::policy::History** (p. 1602)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::History** (p. 1602)

String-version: "QosPolicy.History.KeepAll"

6.137.2.95 snippet_qos_policy_publish_mode_asynchronous()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_publish_mode_asynchronous
( )
```

QoS Snippet that sets PUBLISH_MODE QosPolicy to ASYNCHRONOUS kind.

Modified QoS Parameters:

- **rti::core::policy::PublishMode** (p. 2013)

Asynchronous Publish mode decouples the application thread that calls the **dds::pub::DataWriter** (p. 1138) "write" operation from the thread used to send the data on the network. See https://community.rti.↩ com/glossary/asynchronous-writer

String-version: "QosPolicy.PublishMode.Asynchronous"

6.137.2.96 snippet_qos_policy_durability_transient_local()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_durability_transient↩
_local ( )
```

QoS Snippet that sets DURABILITY QosPolicy to TRANSIENT_LOCAL kind.

Modified QoS Parameters:

- **dds::pub::qos::DataWriterQos** (p. 1226) => **dds::core::policy::Durability** (p. 1433)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::Durability** (p. 1433)

DataWriters will store and send previously published DDS samples for delivery to newly discovered DataReaders as long as the **dds::pub::DataWriter** (p. 1138) still exists. For this setting to be effective, you must also set the **dds::core↩ ::policy::ReliabilityKind_def** (p. 2156) to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) (not Best Effort). Which particular DDS samples are kept depends on other QoS settings such as **dds::core::policy::History** (p. 1602) and **dds::core::policy::ResourceLimits** (p. 2203).

String-version: "QosPolicy.Durability.TransientLocal"

6.137.2.97 snippet_qos_policy_durability_transient()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_durability_transient
( )
```

QoS Snippet that sets DURABILITY QosPolicy to TRANSIENT kind.

Modified QoS Parameters:

- **dds::pub::qos::DataWriterQos** (p. 1226) => **dds::core::policy::Durability** (p. 1433)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::Durability** (p. 1433)

RTI Connext will store previously published DDS samples in memory using Persistence Service, which will send the stored data to newly discovered DataReaders. Which particular DDS samples are kept and sent by Persistence Service depends on the **dds::core::policy::History** (p. 1602) and **dds::core::policy::ResourceLimits** (p. 2203) of the Persistence Service DataWriters. These QosPolicies can be configured in the Persistence Service configuration file or through the **dds::core::policy::DurabilityKind_def** (p. 1441) of the DataWriters configured with **dds::core::policy::↩ DurabilityKind_def::TRANSIENT** (p. 1442).

You need a Persistence Service instance running to use this behavior.

String-version: "QosPolicy.Durability.Transient"

6.137.2.98 snippet_qos_policy_durability_persistent()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_durability_persistent
( )
```

QoS Snippet that sets DURABILITY QosPolicy to PERSISTENT kind.

Modified QoS Parameters:

- **dds::pub::qos::DataWriterQos** (p. 1226) => **dds::core::policy::Durability** (p. 1433)
- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::Durability** (p. 1433)

RTI Connext will store previously published DDS samples in permanent storage, like a disk, using Persistence Service, which will send the stored data to newly discovered DataReaders. Which particular DDS samples are kept and sent by Persistence Service depends on the **dds::core::policy::History** (p. 1602) and **dds::core::policy::ResourceLimits** (p. 2203) in the Persistence Service DataWriters. These QosPolicies can be configured in the Persistence Service configuration file or through the **dds::core::policy::DurabilityKind_def** (p. 1441) of the DataWriters configured with **dds::core::policy::DurabilityKind_def::PERSISTENT** (p. 1442).

You need a Persistence Service instance running to use this behavior.

String-version: "QosPolicy.Durability.Persistent"

6.137.2.99 snippet_qos_policy_batching_enable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_batching_enable ( )
```

QoS Snippet that sets BATCH QosPolicy to true.

Modified QoS Parameters:

- **rti::core::policy::Batch** (p. 880)

This QoS Snippet specifies and configures the mechanism that allows RTI Connext to collect multiple user data DDS samples to be sent in a single network packet, to take advantage of the efficiency of sending larger packets and thus increase effective throughput.

String-version: "QosPolicy.Batching.Enable"

6.137.2.100 snippet_qos_policy_flow_controller_838mbps()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_flow_controller_↩
838mbps ( )
```

QoS Snippet that configures and set a FlowController of 838 Mbps.

Defines a **rti::pub::FlowController** (p. 1572) and configures the **dds::pub::qos::DataWriterQos** (p. 1226) with it.

This is a **rti::pub::FlowController** (p. 1572) of 838 Mbps ($\sim$ 100 MB/sec)

Modified QoS Parameters:

- **dds::domain::qos::DomainParticipantQos** (p. 1384) => **rti::core::policy::Property** (p. 1969) named "dds.↩ flow_controller.token_bucket.$*$"
- **rti::core::policy::PublishMode** (p. 2013)

String-version: "Feature.FlowController.838Mbps"

6.137.2.101 snippet_qos_policy_flow_controller_209mbps()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_flow_controller_↩
209mbps ( )
```

QoS Snippet that configures and sets a FlowController of 209 Mbps.

Defines a **rti::pub::FlowController** (p. 1572) and configures the **dds::pub::qos::DataWriterQos** (p. 1226) with it.

This is a **rti::pub::FlowController** (p. 1572) of 209 Mbps ($\sim$ 25 MB/sec)

Modified QoS Parameters:

- **dds::domain::qos::DomainParticipantQos** (p. 1384) => **rti::core::policy::Property** (p. 1969) named "dds.↩ flow_controller.token_bucket.$*$"
- **rti::core::policy::PublishMode** (p. 2013)

String-version: "Feature.FlowController.209Mbps"

6.137.2.102 snippet_qos_policy_flow_controller_52mbps()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_qos_policy_flow_controller_↩
52mbps ( )
```

QoS Snippet that configures and sets a FlowController of 52 Mbps.

Defines a **rti::pub::FlowController** (p. 1572) and configures the **dds::pub::qos::DataWriterQos** (p. 1226) with it.

This is a **rti::pub::FlowController** (p. 1572) of 52 Mbps ($\sim$ 6.25 MB/sec)

Modified QoS Parameters:

- **dds::domain::qos::DomainParticipantQos** (p. 1384) => **rti::core::policy::Property** (p. 1969) named "dds.↩ flow_controller.token_bucket.$*$"
- **rti::core::policy::PublishMode** (p. 2013)

String-version: "Feature.FlowController.52Mbps"

6.137.2.103 snippet_feature_auto_tuning_enable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_auto_tuning_enable ( )
```

QoS Snippet that enables auto_throttle and turbo_mode to true.

Sets the **dds::domain::qos::DomainParticipantQos** (p. 1384) properties to enable auto_throttle and turbo_mode to true.

Modified QoS Parameters:

- **dds::domain::qos::DomainParticipantQos** (p. 1384) => **rti::core::policy::Property** (p. 1969) named "dds.↩ domain_participant.auto_throttle"
- **dds::pub::qos::DataWriterQos** (p. 1226) => **rti::core::policy::Property** (p. 1969) named "dds.data_writer.↩ auto_throttle.enable"
- **dds::pub::qos::DataWriterQos** (p. 1226) => **rti::core::policy::Property** (p. 1969) named "dds.data_writer.↩ enable_turbo_mode"

The domain_participant.auto_throttle configures the **dds::domain::DomainParticipant** (p. 1326) to gather internal measurements (during **dds::domain::DomainParticipant** (p. 1326) creation) that are required for the Auto Throttle feature. This allows DataWriters belonging to this **dds::domain::DomainParticipant** (p. 1326) to use the Auto Throttle feature.

The turbo_mode adjusts the batch max_data_bytes based on how frequently the **dds::pub::DataWriter** (p. 1138) writes data.

Data_writer.auto_throttle enables automatic throttling in the **dds::pub::DataWriter** (p. 1138) so it can automatically adjust the writing rate and the send window size; this minimizes the need for repairing DDS samples and improves latency.

String-version: "Feature.AutoTuning.Enable"

6.137.2.104 snippet_feature_monitoring_enable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_monitoring_enable ( )
```

QoS Snippet that enables the use of the RTI Monitoring Library.

To enable the use of RTI Monitoring Library apply this QoS Snippet to the QoS Profile used to create your **dds↩ ::domain::DomainParticipant** (p. 1326). For example:

```
 <qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.Monitoring.Enable</element>

    </base_name>

</qos_profile>
```

String-version: "Feature.Monitoring.Enable"

6.137.2.105 snippet_feature_monitoring2_enable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_monitoring2_enable ( )
```

QoS Snippet that enables the use of the RTI Monitoring Library 2.0.

QoS Snippet to enable the use of the RTI Monitoring Library 2.0 with a dedicated DomainParticipant publishing telemetry data from your application in domain ID 2. All metrics are enabled for all resources in this profile.

To enable the use of RTI Monitoring Library 2.0, apply this QoS Snippet to the QoS Profile used to create your DomainParticipantFactory. For example:

```
<qos_profile name="MyProfile" is_default_participant_factory_profile="true">
  <base_name>
    <element>BuiltinQosSnippetLib::Feature.Monitoring2.Enable</element>
  </base_name>
</qos_profile>
```

String-version: "Feature.Monitoring2.Enable"

6.137.2.106 snippet_feature_security_enable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_security_enable ( )
```

QoS Snippet that enables security using the Builtin Security Plugins.

To enable the use of the Builtin DDS Security Library apply this QoS Snippet to the QoS Profile used to create your **dds::domain::DomainParticipant** (p. 1326). For example:

```
 <qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.Security.Enable</element>

    </base_name>

</qos_profile>
```

String-version: "Feature.Security.Enable"

6.137.2.107 snippet_feature_topic_query_enable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_topic_query_enable ( )
```

QoS Snippet that enables Topic Query.

To enable the use of the RTI Connext **rti::sub::TopicQuery** (p. 2516) feature, apply this QoS Snippet to the QoS Profile used to create your **dds::pub::DataWriter** (p. 1138). For example:

```
 <qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.TopicQuery.Enable</element>

    </base_name>

</qos_profile>
```

For more information on Topic Query see the "Topic Queries" chapter in the `User's Manual`.

String-version: "Feature.TopicQuery.Enable"

6.137.2.108 snippet_feature_type_support_v1_enable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_type_support_v1_enable (
)
```

QoS Snippet that enables TypeObject V1 propagation.

To enable TypeObject V1 propagation apply this QoS Snippet to the QoS Profile used to create your **dds::domain::DomainParticipant** (p. 1326). For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.TypeSupport.V1.Enable</element>

    </base_name>

</qos_profile>
```

6.137.2.109 snippet_feature_type_support_v1_disable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_type_support_v1_disable
( )
```

QoS Snippet that disables TypeObject V1 propagation.

To disable TypeObject V1 propagation apply this QoS Snippet to the QoS Profile used to create your **dds::domain::DomainParticipant** (p. 1326). For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.TypeSupport.V1.Disable</element>

    </base_name>

</qos_profile>
```

6.137.2.110 snippet_feature_type_support_v2_enable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_type_support_v2_enable (
)
```

QoS Snippet that enables TypeObject V2 propagation.

To enable TypeObject V2 propagation apply this QoS Snippet to the QoS Profile used to create your **dds::domain::DomainParticipant** (p. 1326). For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.TypeSupport.V2.Enable</element>

    </base_name>

</qos_profile>
```

6.137.2.111 snippet_feature_type_support_v2_disable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_feature_type_support_v2_disable
( )
```

QoS Snippet that disables TypeObject V2 propagation.

To disable TypeObject V2 propagation apply this QoS Snippet to the QoS Profile used to create your **dds::domain::DomainParticipant** (p. 1326). For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.TypeSupport.V2.Disable</element>

    </base_name>

</qos_profile>
```

6.137.2.112 snippet_transport_tcp_lan_client()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_transport_tcp_lan_client ( )
```

QoS Snippet that configures a TCP LAN Client over DDS.

This QoS Snippet sets all the mandatory properties; however the final QoS Profile requires additional configuration.

The values of the initial_peers and the property dds.transport.TCPv4.tcp1.server_bind_port are incorrect (just sample strings). Therefore, they must be modified:

- initial_peers: should point to the remote client IP and port.
- server_bind_port: is the port this application will be using.

This modification should be done in the QoS Profile that will be used to create the **dds::core::Entity** (p. 1520). These new values will overwrite the current invalid values.

String-version: "Transport.TCP.LAN.Client"

6.137.2.113 snippet_transport_tcp_wan_symmetric_client()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_transport_tcp_wan_symmetric_
client ( )
```

QoS Snippet that configures a symmetric WAN TCP Client over DDS.

This QoS Snippet sets all the mandatory properties; however the final QoS Profile requires additional configuration.

The values of the properties dds.transport.TCPv4.tcp1.public_address and dds.transport.TCPv4.tcp1.server_bind_port are incorrect (just sample strings). Also the initial_peers information is not correct. Therefore, the following must be modified:

- initial_peers: should point to the remote client IP and port.
- public_address: public IP address where this application can be reached.
- server_bind_port: port this application will be using.

This modification should be done in the QoS Profile that will be used to create the **dds::core::Entity** (p. 1520). These new values will overwrite the current invalid values.

String-version: "Transport.TCP.WAN.Symmetric.Client"

6.137.2.114 snippet_transport_tcp_wan_asymmetric_server()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_transport_tcp_wan_asymmetric_↩
server ( )
```

QoS Snippet that an asymmetric WAN TCP Server over DDS.

This QoS Snippet sets all the mandatory properties; however the final QoS Profile requires additional configuration.

The values of the properties dds.transport.TCPv4.tcp1.public_address and dds.transport.TCPv4.tcp1.server_bind_port are incorrect (just sample strings). Therefore, they must be modified to the corresponding public_address and port_↩ number. This modification should be done in the QoS Profile that will be used to create the **dds::core::Entity** (p. 1520). These new values will overwrite the current invalid values.

String-version: "Transport.TCP.WAN.Asymmetric.Server"

6.137.2.115 snippet_transport_tcp_wan_asymmetric_client()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_transport_tcp_wan_asymmetric_↩
client ( )
```

QoS Snippet that configures an asymmetric WAN TCP Client over DDS.

This QoS Snippet sets all the mandatory properties; however the final QoS Profile requires additional configuration.

The value of discovery.initial_peers and public_ip have to match the values set on the Server side (**rti::core::builtin_↩ profiles::qos_snippet_lib::snippet_transport_tcp_wan_asymmetric_server()** (p. 504)). This modification should be done in the QoS Profile that will be used to create the **dds::core::Entity** (p. 1520). This new value will overwrite the current invalid value.

String-version: "Transport.TCP.WAN.Asymmetric.Client"

6.137.2.116 snippet_transport_udp_avoid_ip_fragmentation()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_transport_udp_avoid_ip_fragmentation
( )
```

QoS Snippet that configures RTI Connext and the UDP built-in transports (UDPv4, UDPv6, UDPv4_WAN) to avoid IP fragmentation.

For WAN communications and, in general, for communications in third party networks, it is not a good idea to rely on IP fragmentation. IP fragmentation causes significant issues in UDP, where there is no integrated support for a path MTU (maximum transmission unit) discovery protocol as there is in TCP.

This snippet provides a way to avoid IP fragmentation in Connext applications using the built-in UDP transports. Instead, Connext will be responsible for fragmentation, which is done at the RTPS level.

Among other changes, this configuration changes the transport MTU (message_size_max) to be 1400 bytes. Notice that this change will affect other transports such as SHMEM since Connext chooses the minimum transport MTU across all enabled transports to determine the maximum size of outgoing RTPS messages.

String-version: "Transport.UDP.AvoidIPFragmentation"

6.137.2.117 snippet_transport_udp_wan()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_transport_udp_wan ( )
```

QoS Snippet that enables the RTI Real-Time WAN Transport (UDPv4_WAN).

The snippet disables all the other built-in transports and avoids the use of IP fragmentation.

For WAN communications and, in general, for communications in third-party networks, it is not a good idea to rely on IP fragmentation. IP fragmentation causes significant issues in UDP, where there is no integrated support for a path MTU (maximum transmission unit) discovery protocol as there is in TCP.

String-version: "Transport.UDP.WAN"

6.137.2.118 snippet_compatibility_connext_micro_version_2_4_3()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_compatibility_connext_micro_↩
version_2_4_3 ( )
```

QoS Snippet that configures sets LIVELINESS QoS to be compatible with RTI Connext Micro 2.4.3.

QoS Snippet that sets the **dds::sub::qos::DataReaderQos** (p. 1072) and **dds::pub::qos::DataWriterQos** (p. 1226) **dds::core::policy::LivelinessKind_def** (p. 1654) to **dds::core::policy::LivelinessKind_def::MANUAL_BY_TOPIC** (p. 1655) It also disables the built-in shared memory transport.

Modified QoS Parameters:

- **dds::sub::qos::DataReaderQos** (p. 1072) => **dds::core::policy::Liveliness** (p. 1645)
- **dds::pub::qos::DataWriterQos** (p. 1226) => **dds::core::policy::Liveliness** (p. 1645)
- **rti::core::policy::TransportBuiltin** (p. 2533)

RTI Connext Micro versions 2.4.3 and earlier only supported **dds::core::policy::LivelinessKind_def::MANUAL_BY↩_TOPIC** (p. 1655) **dds::core::policy::LivelinessKind_def** (p. 1654). In order to be compatible with RTI Connext Micro 2.4.3, the **dds::sub::DataReader** (p. 976) and **dds::pub::DataWriter** (p. 1138) must have their **dds::core::policy::↩LivelinessKind_def** (p. 1654) changed to this value because the default kind in RTI Connext is **dds::core::policy::↩LivelinessKind_def::AUTOMATIC** (p. 1655).

String-version: "Compatibility.ConnextMicro.Version243"

6.137.2.119 snippet_compatibility_other_dds_vendors_enable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_compatibility_other_dds_vendors↩
_enable ( )
```

QoS Snippet that configures RTI Connext to interoperate with other DDS vendors.

String-version: "Compatibility.OtherDDSVendor.Enable"

6.137.2.120 snippet_compatibility_5_1_0_transport_enable()

```
std::string rti::core::builtin_profiles::qos_snippet_lib::snippet_compatibility_5_1_0_transport_↩
enable ( )
```

QoS Snippet that configures RTI Connext to interoperate with RTI Connext 5.1.0 and below for UDPv6 and SHMEM transports.

String-version: "Compatibility.510Transport.Enable"

6.138 AsyncWaitSet

<<***extension***>> *(*p. *354)* A specialization of **dds::core::cond::WaitSet** (p. 2617) that provides a mechanism to perform the wait asynchronously and uses a thread pool to dispatch the attached active **dds::core::cond::Condition** (p. 948).

Classes

- class **rti::core::cond::AsyncWaitSetProperty**

  Specifies the ***rti::core::cond::AsyncWaitSet*** *(*p. *840) behavior.*
- class **rti::core::cond::AsyncWaitSetCompletionToken**

  Implementation of the completion token role element of the asynchronous completion token pattern that is part of the ***AsyncWaitSet*** *(*p. *840) behavior.*
- class **rti::core::cond::AsyncWaitSet**

  <<***extension***>> *(*p. *354) Dispatches* ***dds::core::cond::Condition*** *(*p. *948) objects using separate threads of execution. Unlike a normal* ***dds::core::cond::WaitSet*** *(*p. *2617), which operates on the current thread, an* ***AsyncWaitSet*** *(*p. *840) uses a thread pool.*
- class **rti::core::cond::AsyncWaitSetListener**

  Listener *(*p. *1637) for receiving event notifications related to the thread pool of the* ***AsyncWaitSet*** *(*p. *840).*
- class **rti::core::cond::NoOpAsyncWaitSetListener**

  A convenience implementation of ***AsyncWaitSetListener*** *(*p. *856) where all methods are overridden to do nothing.*
- class **rti::sub::cond::DataReaderStatusConditionHandler**< **T** >

  Realization of a ***functor handler*** *(*p. *2134) that handles the status of a* ***dds::sub::DataReader*** *(*p. *976).*

Functions

- static **AsyncWaitSetCompletionToken rti::core::cond::AsyncWaitSetCompletionToken::Ignore** ()

  For the operations that allow an ***rti::core::cond::AsyncWaitSetCompletionToken*** *(*p. *853), this sentinel can be provided to indicate an* ***rti::core::cond::AsyncWaitSet*** *(*p. *840) to perform the action associating a 'null' completion token.*

6.138.1 Detailed Description

<<***extension***>> *(*p. *354)* A specialization of **dds::core::cond::WaitSet** (p. 2617) that provides a mechanism to perform the wait asynchronously and uses a thread pool to dispatch the attached active **dds::core::cond::Condition** (p. 948).

This class is a realization of the `Proactor` pattern applied to WaitSets and Conditions that provide a powerful component for your application process events leveraging concurrency.

6.138.2 Function Documentation

6.138.2.1 Ignore()

```
static AsyncWaitSetCompletionToken rti::core::cond::AsyncWaitSetCompletionToken::Ignore ( ) [static]
```

For the operations that allow an **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853), this sentinel can be provided to indicate an **rti::core::cond::AsyncWaitSet** (p. 840) to perform the action associating a 'null' completion token.

This sentinel is a realization of the null object pattern of an **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853). If this object is provided to a **rti::core::cond::AsyncWaitSet** (p. 840) operation, the resulting operation request will be associated with a 'null' completion token that behaves as no-op.

When 'nul' completion token is provided, the **rti::core::cond::AsyncWaitSet** (p. 840) operation returns immediately after it issues the internal request, and there is no mean for your application to wait for the request to complete.

You can use this sentinel when your application can operate on an **rti::core::cond::AsyncWaitSet** (p. 840) without needing to known when operation requests complete or your application uses other strategies to synchronize resources.

This sentinel is also useful when you need to perform operations on **rti::core::cond::AsyncWaitSet** (p. 840) from within one of the threads from its thread pool, where waiting on a valid **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853) is forbidden.

6.139 QoS

Quality of Service (QoS) policies.

Modules

- **All QoS Policies**

  All Quality of Service (QoS) policies.
- **QoS by Category**

  Quality of Service (QoS) organized into categories.

6.139.1 Detailed Description

Quality of Service (QoS) policies.

6.140 All QoS Policies

All Quality of Service (QoS) policies.

Modules

- **QoS Policy Traits**

  *The traits types **policy_id** (p. 1940) and **policy_name** (p. 1941) provide information about a QoS policy.*
- **ASYNCHRONOUS_PUBLISHER**

  *<<**extension**>> (p. 116) Specifies the asynchronous publishing settings of the **dds::pub::Publisher** (p. 1993) instances.*
- **AVAILABILITY**

  *<<**extension**>> (p. 116) Configures the availability of data.*
- **BATCH**

  *<<**extension**>> (p. 116) Batch QoS policy used to enable batching in **dds::pub::DataWriter** (p. 1138) instances.*
- **DATABASE**

  *<<**extension**>> (p. 116) Various threads and resource limits settings used by RTI Connext to control its internal database.*
- **DATA_READER_PROTOCOL**

  *<<**extension**>> (p. 116) Specifies the DataReader-specific protocol QoS.*
- **DATA_READER_RESOURCE_LIMITS**

  *<<**extension**>> (p. 116) Various settings that configure how DataReaders allocate and use physical memory for internal resources.*
- **DATA_REPRESENTATION**

  *<<**spec**>> (p. 112) A list of data representations and compression methods supported by a **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976).*
- **DATA_WRITER_PROTOCOL**

  *<<**extension**>> (p. 116) Along with **rti::core::policy::WireProtocol** (p. 2630) and **rti::core::policy::DataReader↩ Protocol** (p. 1059), this QoS policy configures the DDS on-the-network protocol (RTPS).*
- **DATA_WRITER_RESOURCE_LIMITS**

  *<<**extension**>> (p. 116) Various settings that configure how a **dds::pub::DataWriter** (p. 1138) allocates and uses physical memory for internal resources.*
- **DATA_WRITER_TRANSFER_MODE**

  *<<**extension**>> (p. 116) Specifies the DataWriter transfer mode QoS.*
- **DATA_TAG**

  *<<**spec**>> (p. 112) Stores (name, value) pairs that can be used to determine access permissions.*
- **DEADLINE**

  *<<**spec**>> (p. 112) Expresses the maximum duration (deadline) within which an instance is expected to be updated.*
- **DESTINATION_ORDER**

  *<<**spec**>> (p. 112) Controls the criteria used to determine the logical order among changes made by **dds::pub::↩ Publisher** (p. 1993) entities to the same instance of data (i.e., matching **dds::topic::Topic** (p. 2474) and key).*
- **DISCOVERY**

  *<<**extension**>> (p. 116) Specifies the attributes required to discover participants in the domain.*
- **DISCOVERY_CONFIG**

  *<<**extension**>> (p. 116) Specifies the discovery configuration QoS.*
- **DOMAIN_PARTICIPANT_RESOURCE_LIMITS**

  *<<**extension**>> (p. 116) Various settings that configure how a **dds::domain::DomainParticipant** (p. 1326) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.*
- **DURABILITY**

  *<<**spec**>> (p. 112) This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **dds::sub::DataReader** (p. 976) entities that join the network later.*
- **DURABILITY_SERVICE**

*<<**spec**>> (p. 112) Various settings to configure the external RTI Persistence Service used by RTI Connext for DataWriters with a **dds::core::policy::Durability** (p. 1433) setting of **dds::core::policy::DurabilityKind_def::PERSISTENT** (p. 1442) or **dds::core::policy::DurabilityKind_def::TRANSIENT** (p. 1442).*

- **ENTITY_FACTORY**

  *<<**spec**>> (p. 112) A QoS policy for all **dds::core::Entity** (p. 1520) types that can act as factories for one or more other **dds::core::Entity** (p. 1520) types.*

- **ENTITY_NAME**

  *<<**extension**>> (p. 116) Assigns a name to a **dds::domain::DomainParticipant** (p. 1326), **dds::pub::Publisher** (p. 1993), **dds::sub::Subscriber** (p. 2410), **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976). Except for **dds::pub::Publisher** (p. 1993) and **dds::sub::Subscriber** (p. 2410), these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.*

- **EVENT**

  *<<**extension**>> (p. 116) Configures the internal thread in a DomainParticipant that handles timed events.*

- **HISTORY**

  *<<**spec**>> (p. 112) Specifies the behavior of RTI Connext in the case where the value of an instance changes (one or more times) before it can be successfully communicated to one or more existing subscribers.*

- **GROUP_DATA**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 46) during discovery.*

- **LATENCY_BUDGET**

  *<<**spec**>> (p. 112) Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.*

- **LIFESPAN**

  *<<**spec**>> (p. 112) Specifies how long the data written by the **dds::pub::DataWriter** (p. 1138) is considered valid.*

- **LIVELINESS**

  *<<**spec**>> (p. 112) Specifies and configures the mechanism that allows **dds::sub::DataReader** (p. 976) entities to detect when **dds::pub::DataWriter** (p. 1138) entities become disconnected or "dead."*

- **LOCATORFILTER**

  *<<**extension**>> (p. 116) The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **dds::topic::PublicationBuiltinTopicData** (p. 1977).*

- **LOGGING**

  *<<**extension**>> (p. 116) Configures the RTI Connext logging facility.*

- **MONITORING**

  *<<**extension**>> (p. 116) Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.*

- **MULTICHANNEL**

  *<<**extension**>> (p. 116) Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.*

- **OWNERSHIP**

  *<<**spec**>> (p. 112) Specifies whether it is allowed for multiple **dds::pub::DataWriter** (p. 1138) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.*

- **OWNERSHIP_STRENGTH**

  *<<**spec**>> (p. 112) Specifies the value of the strength used to arbitrate among multiple **dds::pub::DataWriter** (p. 1138) objects that attempt to modify the same instance of a data type (identified by **dds::topic::Topic** (p. 2474) + key).*

- **PARTITION**

  *<<**spec**>> (p. 112) Set of strings that introduces logical partitions in **dds::domain::DomainParticipant** (p. 1326), **dds::pub::Publisher** (p. 1993), or **dds::sub::Subscriber** (p. 2410) entities.*

- **PRESENTATION**

  *<<**spec**>> (p. 112) Specifies how the samples representing changes to data instances are presented to a subscribing application.*

- **PROFILE**

<<***extension***>> (p. 116) Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

- **PROPERTY**

  <<***extension***>> (p. 116) Stores (name, value) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

- **PUBLISH_MODE**

  <<***extension***>> (p. 116) Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its own thread to send data, instead of the user thread.

- **READER_DATA_LIFECYCLE**

  <<***spec***>> (p. 112) Controls how a DataReader manages the lifecycle of the data that it has received.

- **RECEIVER_POOL**

  <<***extension***>> (p. 116) Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

- **RELIABILITY**

  <<***spec***>> (p. 112) Indicates the level of reliability offered/requested by RTI Connext.

- **RESOURCE_LIMITS**

  <<***spec***>> (p. 112) Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

- **SERVICE**

  <<***extension***>> (p. 116) Service QoS policy is used to indicate what kind of service or tool is associated with the DDS entity.

- **SYSTEM_RESOURCE_LIMITS**

  <<***extension***>> (p. 116) Configures DomainParticipant-independent resources used by RTI Connext.

- **TIME_BASED_FILTER**

  <<***spec***>> (p. 112) Filter that allows a ***dds::sub::DataReader*** (p. 976) to specify that it is interested only in (potentially) a subset of the values of the data.

- **TOPIC_DATA**

  Attaches a buffer of opaque data that is distributed by means of ***Built-in Topics*** (p. 46) during discovery.

- **TOPIC_QUERY_DISPATCH**

  <<***extension***>> (p. 116) Configures the ability of a ***dds::pub::DataWriter*** (p. 1138) to publish historical samples

- **TRANSPORT_BUILTIN**

  <<***extension***>> (p. 116) Specifies which built-in transports are used.

- **TRANSPORT_MULTICAST**

  <<***extension***>> (p. 116) Specifies the multicast address on which a ***dds::sub::DataReader*** (p. 976) wants to receive its data. It can also specify a port number, as well as a subset of the available (at the ***dds::domain::DomainParticipant*** (p. 1326) level) transports with which to receive the multicast data.

- **TRANSPORT_MULTICAST_MAPPING**

  <<***extension***>> (p. 116) Specifies a list of topic expressions and addresses that can be used by an Entity with a specific topic name to receive data.

- **TRANSPORT_PRIORITY**

  <<***spec***>> (p. 112) This QoS policy allows the application to take advantage of transport that are capable of sending messages with different priorities.

- **TRANSPORT_SELECTION**

  <<***extension***>> (p. 116) Specifies the physical transports that a ***dds::pub::DataWriter*** (p. 1138) or ***dds::sub::DataReader*** (p. 976) may use to send or receive data.

- **TRANSPORT_UNICAST**

  <<***extension***>> (p. 116) Specifies a subset of transports and a port number that can be used by an Entity to receive data.

- **TYPE_CONSISTENCY_ENFORCEMENT**

*<<**spec**>> (p. 112) Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.*

- **TYPESUPPORT**

  [DEPRECATED] *(p. **296**)* *<<**extension**>> (p. 116) Allows you to attach application-specific values to a **dds::pub::DataWriter*** *(p. 1138) or **dds::sub::DataReader*** *(p. 976), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.*
- **USER_DATA**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics*** *(p. 46) during discovery.*
- **WRITER_DATA_LIFECYCLE**

  *<<**spec**>> (p. 112) Controls how a DataWriter handles the lifecycle of the instances (keys) that it is registered to manage.*
- **WIRE_PROTOCOL**

  *<<**extension**>> (p. 116) Specifies the wire protocol related attributes for the **dds::domain::DomainParticipant*** *(p. 1326).*

Classes

- class **dds::core::policy::QosPolicyCount**

  *<<**value-type**>> (p. 350) Holds a counter for a QosPolicyId*

Typedefs

- typedef std::vector< **QosPolicyCount** > **dds::core::policy::QosPolicyCountSeq**

  *A vector of **QosPolicyCount*** *(p. 2020).*
- typedef uint32_t **dds::core::policy::QosPolicyId**

  Identifies a QoS policy.

6.140.1 Detailed Description

All Quality of Service (QoS) policies.

Data Distribution Service (DDS) relies on the use of QoS. A QoS is a set of characteristics that controls some aspect of the behavior of DDS. A QoS is comprised of individual QoS policies (objects conceptually deriving from an *abstract* `QosPolicy` class).

"Supported QoS policies"

The QosPolicy provides the basic mechanism for an application to specify quality of service parameters. It has an attribute name that is used to uniquely identify each `QosPolicy`.

QosPolicy implementation is comprised of a name, an ID, and a type. The type of a `QosPolicy` value may be atomic, such as an integer or float, or compound (a structure). Compound types are used whenever multiple parameters must be set coherently to define a consistent value for a QosPolicy.

QoS (i.e., a list of `QosPolicy` objects) may be associated with all **dds::core::Entity** (p. 1520) objects in the system such as **dds::topic::Topic** (p. 2474), **dds::pub::DataWriter** (p. 1138), **dds::sub::DataReader** (p. 976), **dds::pub::Publisher** (p. 1993), **dds::sub::Subscriber** (p. 2410), and **dds::domain::DomainParticipant** (p. 1326).

6.140.2 Specifying QoS on entities

QosPolicies can be set programmatically when an **dds::core::Entity** (p. 1520) is created, or modified with the **dds::core::Entity** (p. 1520)'s **set_qos (abstract)** (p. 1522) method.

QosPolicies can also be configured from XML resources (files, strings). With this approach, you can change the QoS without recompiling the application. For more information, see **Configuring QoS Profiles with XML** (p. 302).

To customize a **dds::core::Entity** (p. 1520)'s QoS before creating the entity, the correct pattern is:

- First, initialize a QoS object with the appropriate INITIALIZER constructor.
- Call the relevant get_<entity>_default_qos() method.
- Modify the QoS values as desired.
- Finally, create the entity.

Each `QosPolicy` is treated independently from the others. This approach has the advantage of being very extensible. However, there may be cases where several policies are in conflict. Consistency checking is performed each time the policies are modified via the **set_qos (abstract)** (p. 1522) operation, or when the **dds::core::Entity** (p. 1520) is created.

When a policy is changed after being set to a given value, it is not required that the new value be applied instantaneously; RTI Connext is allowed to apply it after a transition phase. In addition, some `QosPolicy` have immutable semantics, meaning that they can only be specified either at **dds::core::Entity** (p. 1520) creation time or else prior to calling the **dds::core::Entity::enable** (p. 1524) operation on the entity.

Each **dds::core::Entity** (p. 1520) can be configured with a list of `QosPolicy` objects. However, not all QosPolicies are supported by each **dds::core::Entity** (p. 1520). For instance, a **dds::domain::DomainParticipant** (p. 1326) supports a different set of QosPolicies than a **dds::topic::Topic** (p. 2474) or a **dds::pub::Publisher** (p. 1993).

Additional properties that are not exposed through the formal QoS policies can also be set for an **dds::core::Entity** (p. 1520) via the **rti::core::policy::Property** (p. 1969). These properties are described in the `Property Reference Guide`.

6.140.3 QoS compatibility

In several cases, for communications to occur properly (or efficiently), a QosPolicy on the publisher side must be compatible with a corresponding policy on the subscriber side. For example, if a **dds::sub::Subscriber** (p. 2410) requests to receive data reliably while the corresponding **dds::pub::Publisher** (p. 1993) defines a best-effort policy, communication will not happen as requested.

To address this issue and maintain the desirable decoupling of publication and subscription as much as possible, the QosPolicy specification follows the **subscriber-requested, publisher-offered pattern**.

In this pattern, the subscriber side can specify a "requested" value for a particular `QosPolicy`. The publisher side specifes an "offered" value for that `QosPolicy`. RTI Connext will then determine whether the value requested by the subscriber side is compatible with what is offered by the publisher side. If the two policies are compatible, then communication will be established. If the two policies are not compatible, RTI Connext will not establish communications between the two **dds::core::Entity** (p. 1520) objects and will record this fact by means of the **dds::core::status::StatusMask::offered_incompatible_qos()** (p. 2380) on the publisher end and **dds::core::status::StatusMask::requested_incompatible_qos()** (p. 2381) on the subscriber end. The application can detect this fact by means of a **Listener** (p. 1637) or a **dds::core::cond::Condition** (p. 948).

The following **properties** are defined on a QosPolicy.

- **RxO (**p. **514) property**
  The QosPolicy objects that need to be set in a compatible manner between the **publisher** and **subscriber** end are indicated by the setting of the **RxO** (p. 514) property:
  - **RxO** (p. 514) = **YES**
    indicates that the policy can be set both at the publishing and subscribing ends and the values must be set in a compatible manner. In this case the compatible values are explicitly defined.
  - **RxO** (p. 514) = **NO**
    indicates that the policy can be set both at the publishing and subscribing ends but the two settings are independent. That is, all combinations of values are compatible.
  - **RxO** (p. 514) = **N/A**
    indicates that the policy can only be specified at either the publishing or the subscribing end, but not at both ends. So compatibility does not apply.
- **Changeable (**p. **514)property**
  Determines whether a QosPolicy can be changed.
  - **NO (**p. **514)** -- policy can only be specified at **dds::core::Entity** (p. 1520) creation time.
  - **UNTIL ENABLE (**p. **514)** -- policy can only be changed before the **dds::core::Entity** (p. 1520) is enabled.
  - **YES (**p. **514)** -- policy can be changed at any time.

6.140.4 Typedef Documentation

6.140.4.1 QosPolicyCountSeq

```
typedef std::vector< QosPolicyCount>  dds::core::policy::QosPolicyCountSeq
```

A vector of **QosPolicyCount** (p. 2020).

6.140.4.2 QosPolicyId

```
typedef uint32_t  dds::core::policy::QosPolicyId
```

Identifies a QoS policy.

The ID for a given policy can be obtained with **dds::core::policy::policy_id** (p. 1940).

For example, the policy ID for the **dds::core::policy::Deadline** (p. 1255) policy is `dds::core::policy::policy_id<dds::core::policy::Deadline>::value`.

6.141 QoS by Category

Quality of Service (QoS) organized into categories.

Modules

- **Basic QoS**

  QoS that are considered basic and fundamental to RTI Connext.
- **DDS Specification QoS**

  QoS that are defined in one of the OMG DDS Specifications.
- **DDS Extension QoS**

  QoS that are RTI extensions to the DDS standard.
- **Functional Categories**

  *These categories help you to discover which QoS may be relevant or related to a desired functionality or feature. For example, if you are interested in **Sending Large Data** (p. 197), then looking at any QoS in that category will be helpful.*
- **QoS Mutability**

  Some QoS are mutable after an entity is enabled, and others are not.
- **Other QoS Attributes**

  *These attributes are characteristics of the QoS – for example if they are **mutable** (p. 280) or **immutable** (p. 243), or if they **apply per-instance** (p. 287).*
- **Deprecated QoS**

  Deprecated QoS.

6.141.1 Detailed Description

Quality of Service (QoS) organized into categories.

The QoS have been grouped into a number of different categories.

These categories can help you to discover which QoS may be relevant or related to a desired functionality or feature. For example, if you are interested in **Sending Large Data** (p. 197), then looking at any QoS in that category will be helpful. The different functional categories are listed on **Functional Categories** (p. 156).

Some categories are not necessarily related to achieving a specific functionality, but are characteristics of the QoS – for example if they are **mutable** (p. 280) or **immutable** (p. 243), or if they **apply per-instance** (p. 287). These categories are listed under **QoS Mutability** (p. 242) and **Other QoS Attributes** (p. 286).

6.142 ASYNCHRONOUS_PUBLISHER

<<***extension***>> *(p. 116)* Specifies the asynchronous publishing settings of the **dds::pub::Publisher** (p. 1993) instances.

Classes

- class **rti::core::policy::AsynchronousPublisher**

  <<***extension***>> *(p. 116) Configures the mechanism to publish data using a separate thread*

6.142.1 Detailed Description

<<***extension***>> *(*p. *116)* Specifies the asynchronous publishing settings of the **dds::pub::Publisher** (p. 1993) instances.

6.143 AVAILABILITY

<<***extension***>> *(*p. *116)* Configures the availability of data.

Classes

- class **rti::core::EndpointGroup**

  <<***extension***>> *(*p. *354) Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.*
- class **rti::core::policy::Availability**

  <<***extension***>> *(*p. *116) Configures data availability in the context of Collaborative DataWriters and Required Subscriptions*

Typedefs

- typedef **dds::core::vector**< **EndpointGroup** > **rti::core::EndpointGroupSeq**

  *A sequence of **rti::core::EndpointGroup** (*p. *1515).*

6.143.1 Detailed Description

<<***extension***>> *(*p. *116)* Configures the availability of data.

6.143.2 Typedef Documentation

6.143.2.1 EndpointGroupSeq

`typedef` **`dds::core::vector< EndpointGroup> rti::core::EndpointGroupSeq`**

A sequence of **rti::core::EndpointGroup** (p. 1515).

In the context of Collaborative DataWriters, it can be used by a **dds::sub::DataReader** (p. 976) to define a group of remote DataWriters that the **dds::sub::DataReader** (p. 976) will wait to discover before skipping missing samples.

In the context of Durable Subscriptions, it can be used to create a set of Durable Subscriptions identified by a name and a quorum count.

See also

rti::core::EndpointGroup (p. 1515)

6.144 BATCH

<<***extension***>> *(*p. *116)* Batch QoS policy used to enable batching in **dds::pub::DataWriter** (p. 1138) instances.

Classes

- class **rti::core::policy::Batch**

  <<***extension***>> *(*p. *116) Allows a* ***dds::pub::DataWriter*** *(*p. *1138) to batch multiple samples into a single network packet to increase throughput.*

6.144.1 Detailed Description

<<***extension***>> *(*p. *116)* Batch QoS policy used to enable batching in **dds::pub::DataWriter** (p. 1138) instances.

6.145 DATABASE

<<***extension***>> *(*p. *116)* Various threads and resource limits settings used by RTI Connext to control its internal database.

Classes

- class **rti::core::policy::Database**

  <<***extension***>> *(*p. *116) Configures threads and resource limits that RTI Connext uses to control its internal database*

6.145.1 Detailed Description

<<***extension***>> *(*p. *116)* Various threads and resource limits settings used by RTI Connext to control its internal database.

6.146 DATA_READER_PROTOCOL

<<***extension***>> *(*p. *116)* Specifies the DataReader-specific protocol QoS.

Classes

- class **rti::core::policy::DataReaderProtocol**

  <<***extension***>> *(*p. *116) Configures DataReader-specific aspects of the RTPS protocol*
- class **rti::core::RtpsReliableReaderProtocol**

  <<***extension***>> *(*p. *354) Configures aspects of the RTPS protocol related to a reliable DataReader*

6.146.1 Detailed Description

<<***extension***>> *(*p. *116)* Specifies the DataReader-specific protocol QoS.

6.147 DATA_READER_RESOURCE_LIMITS

<<***extension***>> *(*p. *116)* Various settings that configure how DataReaders allocate and use physical memory for internal resources.

Classes

- class **rti::core::policy::DataReaderResourceLimits**

  <<***extension***>> (p. *116) Configures the memory usage of a dds::pub::DataReader*
- struct **rti::core::policy::DataReaderInstanceRemovalKind_def**

  *Sets the kinds of instances that can be replaced when instance resource limits (**dds::core::policy::ResourceLimits**↩ **::max_instances** (p. 2206)) are reached.*
- class **rti::core::DataReaderResourceLimitsInstanceReplacementSettings**

  <<***extension***>> (p. *354) How instances are replaced in the DataReader queue when resource limits are reached.*

Typedefs

- typedef **dds::core::safe_enum**< **DataReaderInstanceRemovalKind_def** > **rti::core::policy::DataReader**↩ **InstanceRemovalKind**

  <<***extension***>> (p. *354)* ***Safe Enumeration*** *(*p. *429) of **DataReaderInstanceRemovalKind_def*** *(*p. *1053)*

Functions

- static DDS_Long **rti::core::policy::DataReaderResourceLimits::auto_max_total_instances** ()

  <<***extension***>> (p. *116) This value is used to make **rti::core::policy::DataReaderResourceLimits::max_total_**↩ **instances** (p. 1096) equal to **dds::core::policy::ResourceLimits::max_instances** (p. 2206).*

6.147.1 Detailed Description

<<***extension***>> *(*p. *116)* Various settings that configure how DataReaders allocate and use physical memory for internal resources.

6.147.2 Typedef Documentation

6.147.2.1 DataReaderInstanceRemovalKind

typedef **dds::core::safe_enum**< **DataReaderInstanceRemovalKind_def**> **rti::core::policy::Data↩ ReaderInstanceRemovalKind**

<<***extension***>> *(*p. *354)* **Safe Enumeration** (p. 429) of **DataReaderInstanceRemovalKind_def** (p. 1053)

See also

DataReaderInstanceRemovalKind_def (p. 1053)

6.147.3 Function Documentation

6.147.3.1 auto_max_total_instances()

static DDS_Long rti::core::policy::DataReaderResourceLimits::auto_max_total_instances ( ) [inline], [static]

<<***extension***>> *(*p. *116)* This value is used to make **rti::core::policy::DataReaderResourceLimits::max_total_↩ instances** (p. 1096) equal to **dds::core::policy::ResourceLimits::max_instances** (p. 2206).

6.148 DATA_REPRESENTATION

<<***spec***>> *(*p. *112)* A list of data representations and compression methods supported by a **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976).

Classes

- class **dds::core::policy::DataRepresentation**

  Contains the data representations supported by entities.
- class **rti::core::CompressionIdMask**

  <<***extension***>> *(*p. *354) Mask that specifies which built-in compression method to used.*
- class **rti::core::CompressionSettings**

  <<***extension***>> *(*p. *354) Compression Settings*

Typedefs

- typedef int16_t **dds::core::policy::DataRepresentationId**

  The type of the elements that ***DataRepresentation*** *(*p. *1113) contains.*
- typedef std::vector< **DataRepresentationId** > **dds::core::policy::DataRepresentationIdSeq**

  A vector of DataRepresentationId.

Functions

- static **DataRepresentationId dds::core::policy::DataRepresentation::xcdr** ()

  Extended Common Data Representation encoding version 1.
- static **DataRepresentationId dds::core::policy::DataRepresentation::xml** ()

  XML Data Representation (unsupported)
- static **DataRepresentationId dds::core::policy::DataRepresentation::xcdr2** ()

  Extended Common Data Representation encoding version 2.
- static **DataRepresentationId dds::core::policy::DataRepresentation::auto_id** ()

  Representation automatically chosen based on the type.

6.148.1 Detailed Description

<<***spec***>> *(p. 112)* A list of data representations and compression methods supported by a **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976).

6.148.2 Typedef Documentation

6.148.2.1 DataRepresentationId

```
typedef int16_t  dds::core::policy::DataRepresentationId
```

The type of the elements that **DataRepresentation** (p. 1113) contains.

2-byte signed integers

6.148.2.2 DataRepresentationIdSeq

```
typedef std::vector< DataRepresentationId>  dds::core::policy::DataRepresentationIdSeq
```

A vector of DataRepresentationId.

6.148.3 Function Documentation

6.148.3.1 xcdr()

```
static  DataRepresentationId dds::core::policy::DataRepresentation::xcdr ( )  [inline], [static]
```

Extended Common Data Representation encoding version 1.

6.148.3.2 xml()

```
static DataRepresentationId dds::core::policy::DataRepresentation::xml ( ) [inline], [static]
```

XML Data Representation (unsupported)

Note

This value is currently not supported.

6.148.3.3 xcdr2()

```
static DataRepresentationId dds::core::policy::DataRepresentation::xcdr2 ( ) [inline], [static]
```

Extended Common Data Representation encoding version 2.

6.148.3.4 auto_id()

```
static DataRepresentationId dds::core::policy::DataRepresentation::auto_id ( ) [inline], [static]
```

Representation automatically chosen based on the type.

For plain language binding, if the allowed_data_representation annotation is not specified or if it contains the value XCDR, RTI Connext translates this field to **dds::core::policy::DataRepresentation::xcdr()** (p. 520). Otherwise, it translates this field to **dds::core::policy::DataRepresentation::xcdr2()** (p. 521).

For the **FlatData language binding** (p. 419), RTI Connext translates this field to **dds::core::policy::Data↩ Representation::xcdr2()** (p. 521).

Examples

Foo.hpp.

6.149 DATA_WRITER_PROTOCOL

<<***extension***>> *(p. 116)* Along with **rti::core::policy::WireProtocol** (p. 2630) and **rti::core::policy::DataReader↩ Protocol** (p. 1059), this QoS policy configures the DDS on-the-network protocol (RTPS).

Classes

- class **rti::core::policy::DataWriterProtocol**

  <<***extension***>> *(p. 116) Configures aspects of an the RTPS protocol related to a **dds::pub::DataWriter** (p. 1138)*

6.149.1 Detailed Description

<<***extension***>> *(p. 116)* Along with **rti::core::policy::WireProtocol** (p. 2630) and **rti::core::policy::DataReader↩ Protocol** (p. 1059), this QoS policy configures the DDS on-the-network protocol (RTPS).

6.150 DATA_WRITER_RESOURCE_LIMITS

<<***extension***>> *(p. 116)* Various settings that configure how a **dds::pub::DataWriter** (p. 1138) allocates and uses physical memory for internal resources.

Classes

- class **rti::core::policy::DataWriterResourceLimits**

  <<***extension***>> *(p. 116) Configures the memory usage of a **dds::pub::DataWriter** (p. 1138)*
- struct **rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_def**

  <<***extension***>> *(p. 354) The enumeration for DataWriter Resource Limits*

Typedefs

- typedef **dds::core::safe_enum< DataWriterResourceLimitsInstanceReplacementKind_def > rti::core↩ ::policy::DataWriterResourceLimitsInstanceReplacementKind**

  <<***extension***>> *(p. 354)* ***Safe Enumeration*** *(p. 429) of **DataWriterResourceLimitsInstanceReplacementKind_def** (p. 1249)*

6.150.1 Detailed Description

<<***extension***>> *(p. 116)* Various settings that configure how a **dds::pub::DataWriter** (p. 1138) allocates and uses physical memory for internal resources.

6.150.2 Typedef Documentation

6.150.2.1 DataWriterResourceLimitsInstanceReplacementKind

```
typedef dds::core::safe_enum< DataWriterResourceLimitsInstanceReplacementKind_def> rti::core↩
::policy::DataWriterResourceLimitsInstanceReplacementKind
```

<<***extension***>> *(p. 354)* **Safe Enumeration** (p. 429) of **DataWriterResourceLimitsInstanceReplacementKind_def** (p. 1249)

See also

DataWriterResourceLimitsInstanceReplacementKind_def (p. 1249)

6.151 DATA_WRITER_TRANSFER_MODE

<<***extension***>> *(*p. *116)* Specifies the DataWriter transfer mode QoS.

Classes

- class **rti::core::policy::DataWriterTransferMode**

  <<***extension***>> *(*p. *116) Configures the transfer mode of a* ***dds::pub::DataWriter*** *(*p. *1138)*
- class **rti::core::DataWriterShmemRefTransferModeSettings**

  <<***extension***>> *(*p. *354) Configures aspects of the shared memory reference transfer mode related to a DataWriter*

6.151.1 Detailed Description

<<***extension***>> *(*p. *116)* Specifies the DataWriter transfer mode QoS.

6.152 DATA_TAG

<<***spec***>> *(*p. *112)* Stores (name, value) pairs that can be used to determine access permissions.

Classes

- class **dds::core::policy::DataTag**

  Stores name-value (string) pairs that can be used to determine access permissions.

6.152.1 Detailed Description

<<***spec***>> *(*p. *112)* Stores (name, value) pairs that can be used to determine access permissions.

The **dds::core::policy::DataTag** (p. 1132) can be used to associate a set of tags in the form of (name, value) pairs with a **dds::sub::DataReader** (p. 976) or **dds::pub::DataWriter** (p. 1138). This is similar to the **rti::core::policy::Property** (p. 1969), except you cannot select whether or not a particular pair should be propagated (included in the built-in topic). Data tags are always propagated. The Access Control plugin may use the tags to determine publish and subscribe permissions.

6.153 DEADLINE

<<***spec***>> *(*p. *112)* Expresses the maximum duration (deadline) within which an instance is expected to be updated.

Classes

- class **dds::core::policy::Deadline**

  Expresses the maximum duration (deadline) within which an instance is expected to be updated.

6.153.1 Detailed Description

<<***spec***>> *(*p. *112)* Expresses the maximum duration (deadline) within which an instance is expected to be updated.

6.154 DESTINATION_ORDER

<<***spec***>> *(*p. *112)* Controls the criteria used to determine the logical order among changes made by **dds::pub::Publisher** (p. 1993) entities to the same instance of data (i.e., matching **dds::topic::Topic** (p. 2474) and key).

Classes

- struct **dds::core::policy::DestinationOrderKind_def**

  The definition of the ***dds::core::safe_enum*** *(*p. *2265) DestinationOrderKind.*
- class **dds::core::policy::DestinationOrder**

  Controls the logical order of updates to the same instance by a ***dds::pub::Publisher*** *(*p. *1993).*
- struct **rti::core::policy::DestinationOrderScopeKind_def**

  <<***extension***>> *(*p. *354) The definition of the* ***dds::core::safe_enum*** *(*p. *2265) DestinationOrderScopeKind*

Typedefs

- typedef **dds::core::safe_enum**< **DestinationOrderKind_def** > **dds::core::policy::DestinationOrderKind**

  Safe Enumeration *(*p. *429) of* ***DestinationOrderKind_def*** *(*p. *1263)*
- typedef **dds::core::safe_enum**< **DestinationOrderScopeKind_def** > **rti::core::policy::DestinationOrderScopeKind**

  Safe Enumeration *(*p. *429) of* ***DestinationOrderScopeKind_def*** *(*p. *1264)*

6.154.1 Detailed Description

<<***spec***>> *(*p. *112)* Controls the criteria used to determine the logical order among changes made by **dds::pub::Publisher** (p. 1993) entities to the same instance of data (i.e., matching **dds::topic::Topic** (p. 2474) and key).

6.154.2 Typedef Documentation

6.154.2.1 DestinationOrderKind

typedef **dds::core::safe_enum< DestinationOrderKind_def> dds::core::policy::DestinationOrderKind**

Safe Enumeration (p. 429) of **DestinationOrderKind_def** (p. 1263)

See also

DestinationOrderKind_def (p. 1263)

6.154.2.2 DestinationOrderScopeKind

typedef **dds::core::safe_enum< DestinationOrderScopeKind_def> rti::core::policy::DestinationOrderScopeKind**

Safe Enumeration (p. 429) of **DestinationOrderScopeKind_def** (p. 1264)

See also

DestinationOrderScopeKind_def (p. 1264)

6.155 DISCOVERY

<<***extension***>> *(*p. *116)* Specifies the attributes required to discover participants in the domain.

Modules

- **NDDS_DISCOVERY_PEERS**

  Environment variable or a file that specifies the default values of ***rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers)*** *(*p. *1268) and* ***rti::core::policy::Discovery::multicast_receive_addresses*** *(*p. *1268) contained in the dds::domain::qos::DomainParticipantQos::discovery qos policy.*

Classes

- class **rti::core::policy::Discovery**

  <<***extension***>> *(*p. *116) Configures entity discovery*

6.155.1 Detailed Description

<<***extension***>> *(*p. *116)* Specifies the attributes required to discover participants in the domain.

6.156 DISCOVERY_CONFIG

<<***extension***>> *(*p. *116)* Specifies the discovery configuration QoS.

Classes

- class **rti::core::policy::BuiltinTopicReaderResourceLimits**

  <<***extension***>> *(*p. *116) Configures several resource management aspects of the built-in topic DataReaders*
- class **rti::core::policy::DiscoveryConfig**

  <<***extension***>> *(*p. *116) Configures the discovery mechanism*
- class **rti::core::policy::DiscoveryConfigBuiltinPluginKindMask**

  <<***extension***>> *(*p. *354) A mask that selects the built-in discovery plugins to be used*
- class **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask**

  <<***extension***>> *(*p. *354) A mask that selects the built-in channels to be used*
- struct **rti::core::policy::RemoteParticipantPurgeKind_def**

  <<***extension***>> *(*p. *354) The definition of the* ***dds::core::safe_enum*** *(*p. *2265) RemoteParticipantPurgeKind*

Typedefs

- typedef **dds::core::safe_enum**< **RemoteParticipantPurgeKind_def** > **rti::core::policy::Remote↩ ParticipantPurgeKind**

  <<***extension***>> *(*p. *354)* ***Safe Enumeration*** *(*p. *429) of* ***RemoteParticipantPurgeKind_def*** *(*p. *2164)*

Functions

- static **DiscoveryConfigBuiltinChannelKindMask rti::core::policy::DiscoveryConfigBuiltinChannelKind↩ Mask::all** ()

  All the bits are set, indicating that all channels are enabled.
- static **DiscoveryConfigBuiltinChannelKindMask rti::core::policy::DiscoveryConfigBuiltinChannelKind↩ Mask::none** ()

  No bits are set, indicating that all channels are disabled.

6.156.1 Detailed Description

<<***extension***>> *(*p. *116)* Specifies the discovery configuration QoS.

6.156.2 Typedef Documentation

6.156.2.1 RemoteParticipantPurgeKind

typedef **dds::core::safe_enum**< **RemoteParticipantPurgeKind_def**> **rti::core::policy::RemoteParticipantPurgeKind**

<<***extension***>> *(*p. *354)* **Safe Enumeration** (p. 429) of **RemoteParticipantPurgeKind_def** (p. 2164)

See also

RemoteParticipantPurgeKind_def (p. 2164)

6.156.3 Function Documentation

6.156.3.1 all()

static **DiscoveryConfigBuiltinChannelKindMask** rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::all ( ) [inline], [static]

All the bits are set, indicating that all channels are enabled.

See also

DiscoveryConfigBuiltinChannelKindMask (p. 1319)

References **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::DiscoveryConfigBuiltinChannelKindMask()**.

6.156.3.2 none()

static **DiscoveryConfigBuiltinChannelKindMask** rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::none ( ) [inline], [static]

No bits are set, indicating that all channels are disabled.

See also

DiscoveryConfigBuiltinChannelKindMask (p. 1319)

References **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::DiscoveryConfigBuiltinChannelKindMask()**.

6.157 DOMAIN_PARTICIPANT_RESOURCE_LIMITS

<<***extension***>> *(p. 116)* Various settings that configure how a **dds::domain::DomainParticipant** (p. 1326) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

Classes

- class **rti::core::policy::DomainParticipantResourceLimits**

  <<***extension***>> *(p. 116) Configures the memory usage of certain* ***dds::domain::DomainParticipant*** *(p. 1326) resources*
- struct **rti::core::policy::IgnoredEntityReplacementKind_def**

  <<***extension***>> *(p. 354) The enumeration for* ***DomainParticipantResourceLimits*** *(p. 1391) ignored entity replacement kinds*
- class **rti::core::AllocationSettings**

  <<***extension***>> *(p. 354) Resource allocation settings*

Typedefs

- typedef **dds::core::safe_enum< IgnoredEntityReplacementKind_def > rti::core::policy::IgnoredEntity↩ ReplacementKind**

  <<***extension***>> *(p. 354)* ***Safe Enumeration*** *(p. 429) of* ***IgnoredEntityReplacementKind_def*** *(p. 1607)*

6.157.1 Detailed Description

<<***extension***>> *(p. 116)* Various settings that configure how a **dds::domain::DomainParticipant** (p. 1326) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

6.157.2 Typedef Documentation

6.157.2.1 IgnoredEntityReplacementKind

```
typedef dds::core::safe_enum< IgnoredEntityReplacementKind_def> rti::core::policy::Ignored↩
EntityReplacementKind
```

<<***extension***>> *(p. 354)* **Safe Enumeration** (p. 429) of **IgnoredEntityReplacementKind_def** (p. 1607)

See also

IgnoredEntityReplacementKind_def (p. 1607)

6.158 DURABILITY

<<***spec***>> *(p. 112)* This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **dds::sub::DataReader** (p. 976) entities that join the network later.

Classes

- struct **dds::core::policy::DurabilityKind_def**

  *The definition of the **dds::core::safe_enum** (p. 2265) DurabilityKind.*
- class **dds::core::policy::Durability**

  *Specifies whether a **dds::pub::DataWriter** (p. 1138) will store and deliver previously published data samples to late-joining **dds::sub::DataReader** (p. 976) entities.*
- class **rti::core::PersistentStorageSettings**

  <<***extension***>> *(p. 354) Configures the persistent storage settings for durable writer history and durable reader state.*

Typedefs

- typedef **dds::core::safe_enum**< **DurabilityKind_def** > **dds::core::policy::DurabilityKind**

  ***Safe Enumeration** (p. 429) of **DurabilityKind_def** (p. 1441)*

Enumerations

- enum class **rti::core::policy::PersistentJournalKind** {
  PersistentJournalKind::delete_journal = DDS_DELETE_PERSISTENT_JOURNAL ,
  PersistentJournalKind::truncate_journal = DDS_TRUNCATE_PERSISTENT_JOURNAL ,
  PersistentJournalKind::persist_journal = DDS_PERSIST_PERSISTENT_JOURNAL ,
  PersistentJournalKind::memory_journal = DDS_MEMORY_PERSISTENT_JOURNAL ,
  PersistentJournalKind::wal_journal = DDS_WAL_PERSISTENT_JOURNAL ,
  PersistentJournalKind::off = DDS_OFF_PERSISTENT_JOURNAL }

  <<***extension***>> *(p. 354) The enumeration for the journal kind of the persistent storage for durable writer history and durable reader state.*
- enum class **rti::core::policy::PersistentSynchronizationKind** {
  PersistentSynchronizationKind::normal = DDS_NORMAL_PERSISTENT_SYNCHRONIZATION ,
  PersistentSynchronizationKind::full = DDS_FULL_PERSISTENT_SYNCHRONIZATION ,
  PersistentSynchronizationKind::off = DDS_OFF_PERSISTENT_SYNCHRONIZATION }

  <<***extension***>> *(p. 354) The enumeration for the synchronization kind of the persistent storage for durable writer history and durable reader state.*

Functions

- int32_t **rti::core::policy::auto_writer_depth** ()

  <<***extension***>> *(p. 116) The default value for **dds::core::policy::Durability::writer_depth** (p. 1439)*

6.158.1 Detailed Description

<<***spec***>> *(p. 112)* This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **dds::sub::DataReader** (p. 976) entities that join the network later.

6.158.2 Typedef Documentation

6.158.2.1 DurabilityKind

`typedef` **`dds::core::safe_enum< DurabilityKind_def> dds::core::policy::DurabilityKind`**

Safe Enumeration (p. 429) of **DurabilityKind_def** (p. 1441)

See also

DurabilityKind_def (p. 1441)

6.158.3 Enumeration Type Documentation

6.158.3.1 PersistentJournalKind

`enum class` **`rti::core::policy::PersistentJournalKind`** `[strong]`

<<***extension***>> *(p. 354)* The enumeration for the journal kind of the persistent storage for durable writer history and durable reader state.

Enumerator

delete_journal	Deletes the rollback journal at the conclusion of each transaction.
truncate_journal	Commits transactions by truncating the rollback journal to zero-length instead of deleting it.
persist_journal	Prevents the rollback journal from being deleted at the end of each transaction. Instead, the header of the journal is overwritten with zeros.
memory_journal	Stores the rollback journal in volatile RAM. This saves disk I/O.
wal_journal	Uses a write-ahead log instead of a rollback journal to implement transactions.
off	Disables the rollback journal.

6.158.3.2 PersistentSynchronizationKind

```
enum class rti::core::policy::PersistentSynchronizationKind [strong]
```

<<***extension***>> *(p. 354)* The enumeration for the synchronization kind of the persistent storage for durable writer history and durable reader state.

Enumerator

normal	Data (e.g., new sample) is written to disk at critical moments.
full	Data (e.g., new sample) is written to physical disk immediately.
off	No synchronization is enforced.

6.158.4 Function Documentation

6.158.4.1 auto_writer_depth()

```
int32_t rti::core::policy::auto_writer_depth ( ) [inline]
```

<<***extension***>> *(p. 116)* The default value for **dds::core::policy::Durability::writer_depth** (p. 1439)

Note

This is a standalone function in the namespace rti::core::policy

This values resolves to the following:

- **dds::core::policy::History::depth** (p. 1605) if the history kind is **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607).
- **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207) in the **dds::core::policy::ResourceLimits** (p. 2203) if the history kind is **dds::core::policy::HistoryKind_def::KEEP_ALL** (p. 1607).

Referenced by **rti::core::policy::DomainParticipantResourceLimits::local_writer_allocation()**.

6.159 DURABILITY_SERVICE

<<***spec***>> *(p. 112)* Various settings to configure the external *RTI Persistence Service* used by RTI Connext for DataWriters with a **dds::core::policy::Durability** (p. 1433) setting of **dds::core::policy::DurabilityKind_def::PERSISTENT** (p. 1442) or **dds::core::policy::DurabilityKind_def::TRANSIENT** (p. 1442).

Classes

- class **dds::core::policy::DurabilityService**

  Configures the external RTI Persistence Service used by persistent and transient DataWriters.

6.159.1 Detailed Description

<<***spec***>> *(*p. *112)* Various settings to configure the external *RTI Persistence Service* used by RTI Connext for DataWriters with a **dds::core::policy::Durability** (p. 1433) setting of **dds::core::policy::DurabilityKind_def::↩ PERSISTENT** (p. 1442) or **dds::core::policy::DurabilityKind_def::TRANSIENT** (p. 1442).

6.160 ENTITY_FACTORY

<<***spec***>> *(*p. *112)* A QoS policy for all **dds::core::Entity** (p. 1520) types that can act as factories for one or more other **dds::core::Entity** (p. 1520) types.

Classes

- class **dds::core::policy::EntityFactory**

  *Configures a **dds::core::Entity** (p. 1520) that acts as factory of other entities.*

6.160.1 Detailed Description

<<***spec***>> *(*p. *112)* A QoS policy for all **dds::core::Entity** (p. 1520) types that can act as factories for one or more other **dds::core::Entity** (p. 1520) types.

6.161 ENTITY_NAME

<<***extension***>> *(*p. *116)* Assigns a name to a **dds::domain::DomainParticipant** (p. 1326), **dds::pub::Publisher** (p. 1993), **dds::sub::Subscriber** (p. 2410), **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976). Except for **dds::pub::Publisher** (p. 1993) and **dds::sub::Subscriber** (p. 2410), these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.

Classes

- class **rti::core::policy::EntityName**

  <<***extension***>> *(*p. *116) Assigns a name to a DomainParticipant, Publisher, Subscriber, DataWriter or DataReader.*

6.161.1 Detailed Description

<<***extension***>> *(p. 116)* Assigns a name to a **dds::domain::DomainParticipant** (p. 1326), **dds::pub::Publisher** (p. 1993), **dds::sub::Subscriber** (p. 2410), **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976). Except for **dds::pub::Publisher** (p. 1993) and **dds::sub::Subscriber** (p. 2410), these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.

6.162 EVENT

<<***extension***>> *(p. 116)* Configures the internal thread in a DomainParticipant that handles timed events.

Classes

- class **rti::core::policy::Event**

  <<***extension***>> *(p. 116) Configures the thread in a DomainParticipant that handles timed events.*

6.162.1 Detailed Description

<<***extension***>> *(p. 116)* Configures the internal thread in a DomainParticipant that handles timed events.

6.163 HISTORY

<<***spec***>> *(p. 112)* Specifies the behavior of RTI Connext in the case where the value of an instance changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

Classes

- struct **dds::core::policy::HistoryKind_def**

  *The definition of the **dds::core::safe_enum** (p. 2265) HistoryKind.*
- class **dds::core::policy::History**

  *Specifies how much historical data a **dds::pub::DataWriter** (p. 1138) and a **dds::sub::DataReader** (p. 976) can store.*

Typedefs

- typedef **dds::core::safe_enum**< **HistoryKind_def** > **dds::core::policy::HistoryKind**

  ***Safe Enumeration** (p. 429) of **HistoryKind_def** (p. 1606)*

6.163.1 Detailed Description

<<***spec***>> *(p. 112)* Specifies the behavior of RTI Connext in the case where the value of an instance changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

6.163.2 Typedef Documentation

6.163.2.1 HistoryKind

`typedef` **`dds::core::safe_enum< HistoryKind_def> dds::core::policy::HistoryKind`**

Safe Enumeration (p. 429) of **HistoryKind_def** (p. 1606)

See also

HistoryKind_def (p. 1606)

6.164 GROUP_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 46) during discovery.

Classes

- class **dds::core::policy::GroupData**

6.164.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 46) during discovery.

6.165 LATENCY_BUDGET

<<***spec***>> *(p. 112)* Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

Classes

- class **dds::core::policy::LatencyBudget**

  Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

6.165.1 Detailed Description

<<***spec***>> *(p. 112)* Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

6.166 LIFESPAN

<<***spec***>> *(p. 112)* Specifies how long the data written by the **dds::pub::DataWriter** (p. 1138) is considered valid.

Classes

- class **dds::core::policy::Lifespan**

  *Specifies how long the data written by a **dds::pub::DataWriter** (p. 1138) is considered valid.*

6.166.1 Detailed Description

<<***spec***>> *(p. 112)* Specifies how long the data written by the **dds::pub::DataWriter** (p. 1138) is considered valid.

6.167 LIVELINESS

<<***spec***>> *(p. 112)* Specifies and configures the mechanism that allows **dds::sub::DataReader** (p. 976) entities to detect when **dds::pub::DataWriter** (p. 1138) entities become disconnected or "dead."

Classes

- struct **dds::core::policy::LivelinessKind_def**

  *The definition of the **dds::core::safe_enum** (p. 2265) LivelinessKind.*
- class **dds::core::policy::Liveliness**

  *Specifies and configures the mechanism that allows **dds::sub::DataReader** (p. 976)'s to detect when **dds::pub::Data↩ Writer** (p. 1138)'s become disconnected.*

Typedefs

- typedef **dds::core::safe_enum**< **LivelinessKind_def** > **dds::core::policy::LivelinessKind**

  ***Safe Enumeration** (p. 429) of **LivelinessKind_def** (p. 1654)*

6.167.1 Detailed Description

<<***spec***>> *(p. 112)* Specifies and configures the mechanism that allows **dds::sub::DataReader** (p. 976) entities to detect when **dds::pub::DataWriter** (p. 1138) entities become disconnected or "dead."

6.167.2 Typedef Documentation

6.167.2.1 LivelinessKind

```
typedef dds::core::safe_enum< LivelinessKind_def> dds::core::policy::LivelinessKind
```

Safe Enumeration (p. 429) of **LivelinessKind_def** (p. 1654)

See also

LivelinessKind_def (p. 1654)

6.168 LOCATORFILTER

<<***extension***>> *(p. 116)* The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **dds::topic::PublicationBuiltinTopicData** (p. 1977).

Classes

- class **rti::core::LocatorFilterElement**

  <<***extension***>> (p. *354)* <<***value-type***>> (p. *350) Specifies the configuration of an individual channel within a Multi↩ Channel DataWriter.*
- class **rti::core::policy::LocatorFilter**

  <<***extension***>> (p. *116) Configures how the* ***dds::topic::PublicationBuiltinTopicData*** (p. *1977) reports the configuration of a* ***MultiChannel*** (p. *1749) DataWriter.*

6.168.1 Detailed Description

<<***extension***>> *(p. 116)* The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **dds::topic::PublicationBuiltinTopicData** (p. 1977).

6.169 LOGGING

<<***extension***>> *(p. 116)* Configures the RTI Connext logging facility.

<<***extension***>> *(p. 116)* Configures the RTI Connext logging facility.

6.170 MONITORING

<<***extension***>> *(p. 116)* Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.

Classes

- class **rti::core::policy::Monitoring**

  *<<**extension**>> (p. 116) Configures the use of the RTI **Monitoring** (p. 1706) Library 2.0 to collect and distribute RTI Connext telemetry data.*
- class **rti::core::MonitoringPeriodicDistributionSettings**

  *<<**extension**>> (p. 354) Configures the distribution of periodic metrics.*
- class **rti::core::MonitoringEventDistributionSettings**

  *<<**extension**>> (p. 354) Configures the distribution of event metrics.*
- class **rti::core::MonitoringLoggingDistributionSettings**

  *<<**extension**>> (p. 354) Configures the distribution of log messages.*
- class **rti::core::MonitoringDedicatedParticipantSettings**

  *<<**extension**>> (p. 354) Configures the usage of a dedicated **dds::domain::DomainParticipant** (p. 1326) to distribute the RTI Connext application telemetry data.*
- class **rti::core::MonitoringDistributionSettings**

  *<<**extension**>> (p. 354) Configures the distribution of telemetry data.*
- class **rti::core::MonitoringMetricSelection**

  *<<**extension**>> (p. 354) Configures event and periodic metrics collection and distribution for a specific set of observable resources.*
- class **rti::core::MonitoringLoggingForwardingSettings**

  *<<**extension**>> (p. 354) Configures the forwarding levels of log messages for the different **rti::config::LogFacility** (p. 459).*
- class **rti::core::MonitoringTelemetryData**

  *<<**extension**>> (p. 354) Configures the telemetry data that will be distributed.*

6.170.1 Detailed Description

*<<**extension**>> (p. 116)* Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.

6.171 MULTICHANNEL

*<<**extension**>> (p. 116)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

Classes

- class **rti::core::policy::MultiChannel**

  *<<**extension**>> (p. 116) Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.*
- class **rti::core::ChannelSettings**

  *<<**extension**>> (p. 354) Configures the properties of a channel in **rti::core::policy::MultiChannel** (p. 1749)*

Typedefs

- typedef std::vector< **ChannelSettings** > **rti::core::ChannelSettingsSeq**

  <<***extension***>> *(p. 354) A sequence of* ***ChannelSettings*** *(p. 921)*

6.171.1 Detailed Description

<<***extension***>> *(p. 116)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

6.171.2 Typedef Documentation

6.171.2.1 ChannelSettingsSeq

```
typedef std::vector< ChannelSettings>  rti::core::ChannelSettingsSeq
```

<<***extension***>> *(p. 354)* A sequence of **ChannelSettings** (p. 921)

6.172 OWNERSHIP

<<***spec***>> *(p. 112)* Specifies whether it is allowed for multiple **dds::pub::DataWriter** (p. 1138) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

Classes

- struct **dds::core::policy::OwnershipKind_def**

  The definition of the ***dds::core::safe_enum*** *(p. 2265) OwnershipKind.*
- class **dds::core::policy::Ownership**

  Specifies whether it is allowed for multiple ***dds::pub::DataWriter*** *(p. 1138)'s to write the same instance of the data and if so, how these modifications should be arbitrated.*

Typedefs

- typedef **dds::core::safe_enum**< **OwnershipKind_def** > **dds::core::policy::OwnershipKind**

  Safe Enumeration *(p. 429) of* ***OwnershipKind_def*** *(p. 1905)*

6.172.1 Detailed Description

<<***spec***>> *(p. 112)* Specifies whether it is allowed for multiple **dds::pub::DataWriter** (p. 1138) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

6.172.2 Typedef Documentation

6.172.2.1 OwnershipKind

`typedef` **`dds::core::safe_enum< OwnershipKind_def> dds::core::policy::OwnershipKind`**

Safe Enumeration (p. 429) of **OwnershipKind_def** (p. 1905)

See also

OwnershipKind_def (p. 1905)

6.173 OWNERSHIP_STRENGTH

<<***spec***>> *(p. 112)* Specifies the value of the strength used to arbitrate among multiple **dds::pub::DataWriter** (p. 1138) objects that attempt to modify the same instance of a data type (identified by **dds::topic::Topic** (p. 2474) + key).

Classes

- class **dds::core::policy::OwnershipStrength**

  *Specifies the value of the strength used to arbitrate among multiple **dds::pub::DataWriter** (p. 1138) objects that attempt to modify the same instance of a data type (identified by its **dds::topic::Topic** (p. 2474) and key).*

6.173.1 Detailed Description

<<***spec***>> *(p. 112)* Specifies the value of the strength used to arbitrate among multiple **dds::pub::DataWriter** (p. 1138) objects that attempt to modify the same instance of a data type (identified by **dds::topic::Topic** (p. 2474) + key).

6.174 PARTITION

<<***spec***>> *(p. 112)* Set of strings that introduces logical partitions in **dds::domain::DomainParticipant** (p. 1326), **dds::pub::Publisher** (p. 1993), or **dds::sub::Subscriber** (p. 2410) entities.

Classes

- class **dds::core::policy::Partition**

  *Set of strings that introduces logical partitions in **dds::domain::DomainParticipant** (p. 1326), **dds::pub::Publisher** (p. 1993), or **dds::sub::Subscriber** (p. 2410) entities.*

6.174.1 Detailed Description

<<***spec***>> *(p. 112)* Set of strings that introduces logical partitions in **dds::domain::DomainParticipant** (p. 1326), **dds::pub::Publisher** (p. 1993), or **dds::sub::Subscriber** (p. 2410) entities.

6.175 PRESENTATION

<<***spec***>> *(p. 112)* Specifies how the samples representing changes to data instances are presented to a subscribing application.

Classes

- struct **dds::core::policy::PresentationAccessScopeKind_def**

  *The definition of the **dds::core::safe_enum** (p. 2265) PresentationAccessScopeKind.*
- class **dds::core::policy::Presentation**

  Specifies how the samples representing changes to data instances are presented to a subscribing application.

Typedefs

- typedef **dds::core::safe_enum< PresentationAccessScopeKind_def > dds::core::policy::Presentation↩ AccessScopeKind**

  ***Safe Enumeration** (p. 429) of **PresentationAccessScopeKind_def** (p. 1950)*

6.175.1 Detailed Description

<<***spec***>> *(p. 112)* Specifies how the samples representing changes to data instances are presented to a subscribing application.

6.175.2 Typedef Documentation

6.175.2.1 PresentationAccessScopeKind

`typedef` **`dds::core::safe_enum< PresentationAccessScopeKind_def> dds::core::policy::Presentation↩ AccessScopeKind`**

Safe Enumeration (p. 429) of **PresentationAccessScopeKind_def** (p. 1950)

See also

PresentationAccessScopeKind_def (p. 1950)

6.176 PROFILE

<<***extension***>> *(p. 116)* Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

Classes

- class **rti::core::QosProviderParams**

  <<***extension***>> (p. *354)* <<***value-type***>> (p. *350) Configure options that control the way that XML documents containing QoS profiles are loaded by a* ***dds::core::QosProvider*** *(p. 2025).*

6.176.1 Detailed Description

<<***extension***>> *(p. 116)* Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

6.177 PROPERTY

<<***extension***>> *(p. 116)* Stores (name, value) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

Classes

- class **rti::core::policy::Property**

  <<***extension***>> (p. *116) Stores key/value string pairs that can configure certain parameters of RTI Connext not exposed through QoS policies. It can also store and propagate through the discovery mechanism application-specific information associated to a* ***dds::core::Entity*** *(p. 1520).*

6.177.1 Detailed Description

<<***extension***>> *(p. 116)* Stores (name, value) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

RTI Connext will automatically set some system properties in the **rti::core::policy::Property** (p. 1969) associated with a **dds::domain::qos::DomainParticipantQos** (p. 1384). See **System Properties** (p. 301) for additional details.

6.178 PUBLISH_MODE

<<***extension***>> *(p. 116)* Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its *own* thread to send data, instead of the user thread.

Classes

- class **rti::core::policy::PublishMode**

  <<***extension***>> *(p. 116) Specifies whether a* ***dds::pub::DataWriter*** *(p. 1138) sends data synchronously or asynchronously.*
- struct **rti::core::policy::PublishModeKind_def**

  <<***extension***>> *(p. 354) The enumeration for* ***PublishMode*** *(p. 2013) kinds*

Typedefs

- typedef **dds::core::safe_enum< PublishModeKind_def > rti::core::policy::PublishModeKind**

  <<***extension***>> *(p. 354)* ***Safe Enumeration*** *(p. 429) of* ***PublishModeKind_def*** *(p. 2018)*

Variables

- const int32_t **rti::core::PUBLICATION_PRIORITY_UNDEFINED**

  Initializer value for ***rti::core::policy::PublishMode::priority*** *(p. 2017) and/or* ***rti::core::ChannelSettings::priority*** *(p. 924).*
- const int32_t **rti::core::PUBLICATION_PRIORITY_AUTOMATIC**

  Constant value for ***rti::core::policy::PublishMode::priority*** *(p. 2017) and/or* ***rti::core::ChannelSettings::priority*** *(p. 924).*

6.178.1 Detailed Description

<<***extension***>> *(p. 116)* Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its *own* thread to send data, instead of the user thread.

6.178.2 Typedef Documentation

6.178.2.1 PublishModeKind

```
typedef dds::core::safe_enum< PublishModeKind_def> rti::core::policy::PublishModeKind
```

<<***extension***>> *(p. 354)* **Safe Enumeration** (p. 429) of **PublishModeKind_def** (p. 2018)

See also

PublishModeKind_def (p. 2018)

6.178.3 Variable Documentation

6.178.3.1 PUBLICATION_PRIORITY_UNDEFINED

```
const int32_t rti::core::PUBLICATION_PRIORITY_UNDEFINED
```

Initializer value for **rti::core::policy::PublishMode::priority** (p. 2017) and/or **rti::core::ChannelSettings::priority** (p. 924).

When assigned this value, the publication priority of the data writer, or channel of a multi-channel data writer, will be set to the lowest possible value. For muilti-channel data writers, if either the data writer or channel priority is NOT set to this value, then the publication priority of the entity will be set to the defined value.

6.178.3.2 PUBLICATION_PRIORITY_AUTOMATIC

```
const int32_t rti::core::PUBLICATION_PRIORITY_AUTOMATIC
```

Constant value for **rti::core::policy::PublishMode::priority** (p. 2017) and/or **rti::core::ChannelSettings::priority** (p. 924).

When assigned this value, the publication priority of the data writer, or channel of a multi-channel data writer, will be set to the largest priority value of any sample currently queued for publication by the data writer or data writer channel.

6.179 READER_DATA_LIFECYCLE

<<***spec***>> *(p. 112)* Controls how a DataReader manages the lifecycle of the data that it has received.

Classes

- class **dds::core::policy::ReaderDataLifecycle**

  Controls how a DataReader manages the lifecycle of the data that it has received.

6.179.1 Detailed Description

<<***spec***>> *(p. 112)* Controls how a DataReader manages the lifecycle of the data that it has received.

6.180 RECEIVER_POOL

<<***extension***>> *(p. 116)* Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

Classes

- class **rti::core::policy::ReceiverPool**

  <<***extension***>> *(*p. *116) Configures threads that RTI Connext uses to receive and process data from the transport modules (such as UDP)*

6.180.1 Detailed Description

<<***extension***>> *(*p. *116)* Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

6.181 RELIABILITY

<<***spec***>> *(*p. *112)* Indicates the level of reliability offered/requested by RTI Connext.

Classes

- struct **dds::core::policy::ReliabilityKind_def**

  The definition of the ***dds::core::safe_enum*** *(*p. *2265) ReliabilityKind.*
- class **dds::core::policy::Reliability**

  Indicates the level of reliability in sample delivered that a ***dds::pub::DataWriter*** *(*p. *1138) offers or a* ***dds::sub::Data↩ Reader*** *(*p. *976) requests.*
- struct **rti::core::policy::AcknowledgmentKind_def**

  <<***extension***>> *(*p. *354) The enumeration for Reliability acknowledgment kinds*

Typedefs

- typedef **dds::core::safe_enum< ReliabilityKind_def > dds::core::policy::ReliabilityKind**

  Safe Enumeration *(*p. *429) of* ***ReliabilityKind_def*** *(*p. *2156)*
- typedef **dds::core::safe_enum< AcknowledgmentKind_def > rti::core::policy::AcknowledgmentKind**

  <<***extension***>> *(*p. *354)* ***Safe Enumeration*** *(*p. *429) of* ***AcknowledgmentKind_def*** *(*p. *797)*

Enumerations

- enum class **rti::core::policy::InstanceStateConsistencyKind** {
  InstanceStateConsistencyKind::none ,
  InstanceStateConsistencyKind::recover_state }

  <<***extension***>> *(*p. *354) The enumeration for instance state consistency kinds*

6.181.1 Detailed Description

<<***spec***>> *(*p. *112)* Indicates the level of reliability offered/requested by RTI Connext.

6.181.2 Typedef Documentation

6.181.2.1 ReliabilityKind

typedef **dds::core::safe_enum**< **ReliabilityKind_def**> **dds::core::policy::ReliabilityKind**

Safe Enumeration (p. 429) of **ReliabilityKind_def** (p. 2156)

See also

ReliabilityKind_def (p. 2156)

6.181.2.2 AcknowledgmentKind

typedef **dds::core::safe_enum**< **AcknowledgmentKind_def**> **rti::core::policy::AcknowledgmentKind**

<<***extension***>> *(p. 354)* **Safe Enumeration** (p. 429) of **AcknowledgmentKind_def** (p. 797)

See also

AcknowledgmentKind_def (p. 797)

6.181.3 Enumeration Type Documentation

6.181.3.1 InstanceStateConsistencyKind

enum class **rti::core::policy::InstanceStateConsistencyKind** [strong]

<<***extension***>> *(p. 354)* The enumeration for instance state consistency kinds

Enumerator

none	Instance states on the DataReader are not guaranteed to be correct after liveliness is regained after a disconnect. When DataReaders rediscover DataWriters, they will not request updated instance state data. DataWriters always provide instance state data alongside each sample update regardless of this setting.
recover_state	Instance states on the DataReader are guaranteed to be correct after liveliness is regained after a disconnect. When DataReaders rediscover DataWriters, they will request updated instance state data. DataWriters will respond to requests for updated instance state data and publish updates on a builtin channel. DataWriters still provide instance state data alongside each sample update regardless of this setting. The following limitation applies to using RECOVER_INSTANCE_STATE_CONSISTENCY: If the **dds::core::policy::DestinationOrderKind_def** (p. 1263) is set to **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264),

Generated by Doxygen

6.182 RESOURCE_LIMITS

<<***spec***>> *(*p. *112)* Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

Classes

- class **dds::core::policy::ResourceLimits**

  Controls the memory usage of ***dds::pub::DataWriter*** *(*p. *1138) or a* ***dds::sub::DataReader*** *(*p. *976).*

6.182.1 Detailed Description

<<***spec***>> *(*p. *112)* Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

6.183 SERVICE

<<***extension***>> *(*p. *116)* Service QoS policy is used to indicate what kind of service or tool is associated with the DDS entity.

Classes

- class **rti::core::policy::Service**

  <<***extension***>> *(*p. *116) Indicates if an Entity is associated with a service or tool and if so, which one.*
- struct **rti::core::policy::ServiceKind_def**

  <<***extension***>> *(*p. *354) The definition of the* ***dds::core::safe_enum*** *(*p. *2265) ServiceKind*

Typedefs

- typedef **dds::core::safe_enum**< **ServiceKind_def** > **rti::core::policy::ServiceKind**

  <<***extension***>> *(*p. *354)* ***Safe Enumeration*** *(*p. *429) of* ***ServiceKind_def*** *(*p. *2357)*

6.183.1 Detailed Description

<<***extension***>> *(*p. *116)* Service QoS policy is used to indicate what kind of service or tool is associated with the DDS entity.

6.183.2 Typedef Documentation

6.183.2.1 ServiceKind

```
typedef dds::core::safe_enum< ServiceKind_def> rti::core::policy::ServiceKind
```

<<***extension***>> *(*p. *354)* **Safe Enumeration** (p. 429) of **ServiceKind_def** (p. 2357)

See also

ServiceKind_def (p. 2357)

6.184 SYSTEM_RESOURCE_LIMITS

<<***extension***>> *(*p. *116)* Configures DomainParticipant-independent resources used by RTI Connext.

Classes

- class **rti::core::policy::SystemResourceLimits**
  <<***extension***>> *(*p. *116) Configures resources that RTI Connext uses*

6.184.1 Detailed Description

<<***extension***>> *(*p. *116)* Configures DomainParticipant-independent resources used by RTI Connext.

6.185 TIME_BASED_FILTER

<<***spec***>> *(*p. *112)* Filter that allows a **dds::sub::DataReader** (p. 976) to specify that it is interested only in (potentially) a subset of the values of the data.

Classes

- class **dds::core::policy::TimeBasedFilter**
  *Allows a **dds::sub::DataReader** (p. 976) to indicate that it is not interested in all the sample updates that occur within a time period.*

6.185.1 Detailed Description

<<***spec***>> *(*p. *112)* Filter that allows a **dds::sub::DataReader** (p. 976) to specify that it is interested only in (potentially) a subset of the values of the data.

6.186 TOPIC_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 46) during discovery.

Classes

- class **dds::core::policy::TopicData**

6.186.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 46) during discovery.

6.187 TOPIC_QUERY_DISPATCH

<<***extension***>> *(p. 116)* Configures the ability of a **dds::pub::DataWriter** (p. 1138) to publish historical samples

Classes

- class **rti::core::policy::TopicQueryDispatch**

  <<***extension***>> *(p. 116) Configures the ability of a **dds::pub::DataWriter** (p. 1138) to publish samples in response to a **rti::sub::TopicQuery** (p. 2516)*

6.187.1 Detailed Description

<<***extension***>> *(p. 116)* Configures the ability of a **dds::pub::DataWriter** (p. 1138) to publish historical samples

6.188 TRANSPORT_BUILTIN

<<***extension***>> *(p. 116)* Specifies which built-in transports are used.

Classes

- class **rti::core::policy::TransportBuiltin**

  <<***extension***>> *(p. 116) Specifies which built-in transports to use*
- class **rti::core::policy::TransportBuiltinMask**

  <<***extension***>> *(p. 354) Mask that specifies which built-in transports are used*

6.188.1 Detailed Description

<<***extension***>> *(*p. *116)* Specifies which built-in transports are used.

6.189 TRANSPORT_MULTICAST

<<***extension***>> *(*p. *116)* Specifies the multicast address on which a **dds::sub::DataReader** (p. 976) wants to receive its data. It can also specify a port number, as well as a subset of the available (at the **dds::domain::DomainParticipant** (p. 1326) level) transports with which to receive the multicast data.

Classes

- class **rti::core::policy::TransportMulticast**

  <<***extension***>> *(*p. *116) Specifies the multicast address on which a **dds::sub::DataReader** (*p. *976) wants to receive its data and other settings.*
- struct **rti::core::policy::TransportMulticastKind_def**

  <<***extension***>> *(*p. *354) The definition of the **dds::core::safe_enum** (*p. *2265) TransportMulticastKind*
- class **rti::core::TransportMulticastSettings**

  <<***extension***>> *(*p. *354) Represents a list of multicast locators*

Typedefs

- typedef **dds::core::safe_enum**< **TransportMulticastKind_def** > **rti::core::policy::TransportMulticastKind**

  <<***extension***>> *(*p. *354) **Safe Enumeration** (*p. *429) of **TransportMulticastKind_def** (*p. *2547)*
- typedef std::vector< **TransportMulticastSettings** > **rti::core::TransportMulticastSettingsSeq**

  *A sequence of **TransportMulticastSettings** (*p. *2549).*

6.189.1 Detailed Description

<<***extension***>> *(*p. *116)* Specifies the multicast address on which a **dds::sub::DataReader** (p. 976) wants to receive its data. It can also specify a port number, as well as a subset of the available (at the **dds::domain::DomainParticipant** (p. 1326) level) transports with which to receive the multicast data.

6.189.2 Typedef Documentation

6.189.2.1 TransportMulticastKind

```
typedef dds::core::safe_enum< TransportMulticastKind_def> rti::core::policy::TransportMulticast↩
Kind
```

<<***extension***>> *(*p. *354)* **Safe Enumeration** (p. 429) of **TransportMulticastKind_def** (p. 2547)

See also

TransportMulticastKind_def (p. 2547)

6.189.2.2 TransportMulticastSettingsSeq

```
typedef std::vector< TransportMulticastSettings> rti::core::TransportMulticastSettingsSeq
```

A sequence of **TransportMulticastSettings** (p. 2549).

6.190 TRANSPORT_MULTICAST_MAPPING

<<***extension***>> *(*p. *116)* Specifies a list of topic expressions and addresses that can be used by an Entity with a specific topic name to receive data.

Classes

- class **rti::core::policy::TransportMulticastMapping**

  <<***extension***>> *(*p. *116) Specifies a list of topic_expressions and multicast addresses that can be used by an Entity with a specific topic name to receive data.*

6.190.1 Detailed Description

<<***extension***>> *(*p. *116)* Specifies a list of topic expressions and addresses that can be used by an Entity with a specific topic name to receive data.

6.191 TRANSPORT_PRIORITY

<<***spec***>> *(*p. *112)* This QoS policy allows the application to take advantage of transport that are capable of sending messages with different priorities.

Classes

- class **dds::core::policy::TransportPriority**

  Allows applications to take advantage of transports capable of sending messages with different priorities.

6.191.1 Detailed Description

<<***spec***>> *(*p. *112)* This QoS policy allows the application to take advantage of transport that are capable of sending messages with different priorities.

6.192 TRANSPORT_SELECTION

<<***extension***>> *(*p. *116)* Specifies the physical transports that a **dds::pub::DataWriter** (p. 1138) or **dds::sub::↩ DataReader** (p. 976) may use to send or receive data.

Classes

- class **rti::core::policy::TransportSelection**

  <<***extension***>> *(*p. *116) Specifies the transports that a **dds::pub::DataWriter** (*p. *1138) or a **dds::sub::DataReader** (*p. *976) may use to send or receive data*

6.192.1 Detailed Description

<<***extension***>> *(*p. *116)* Specifies the physical transports that a **dds::pub::DataWriter** (p. 1138) or **dds::sub::↩ DataReader** (p. 976) may use to send or receive data.

6.193 TRANSPORT_UNICAST

<<***extension***>> *(*p. *116)* Specifies a subset of transports and a port number that can be used by an Entity to receive data.

Classes

- class **rti::core::policy::TransportUnicast**

  <<***extension***>> *(*p. *116) Specifies a subset of transports and a port number that can be used by a **dds::core::Entity** (*p. *1520) to receive data.*
- class **rti::core::TransportUnicastSettings**

  <<***extension***>> (p. *354) Represents a list of unicast locators*

Typedefs

- typedef std::vector< **TransportUnicastSettings** > **rti::core::TransportUnicastSettingsSeq**

  <<***extension***>> *(p. 354) A sequence of* ***TransportUnicastSettings*** *(p. 2559)*

6.193.1 Detailed Description

<<***extension***>> *(p. 116)* Specifies a subset of transports and a port number that can be used by an Entity to receive data.

6.193.2 Typedef Documentation

6.193.2.1 TransportUnicastSettingsSeq

```
typedef std::vector< TransportUnicastSettings>  rti::core::TransportUnicastSettingsSeq
```

<<***extension***>> *(p. 354)* A sequence of **TransportUnicastSettings** (p. 2559)

6.194 TYPE_CONSISTENCY_ENFORCEMENT

<<***spec***>> *(p. 112)* Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

Classes

- struct **dds::core::policy::TypeConsistencyEnforcementKind_def**

  The definition of the ***dds::core::safe_enum*** *(p. 2265) TypeConsistencyEnforcementKind.*
- class **dds::core::policy::TypeConsistencyEnforcement**

  Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

Typedefs

- typedef **dds::core::safe_enum**< **TypeConsistencyEnforcementKind_def** > **dds::core::policy::Type↩ ConsistencyEnforcementKind**

  Safe Enumeration *(p. 429) of* ***TypeConsistencyEnforcementKind_def*** *(p. 2570)*

6.194.1 Detailed Description

<<***spec***>> *(p. 112)* Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

6.194.2 Typedef Documentation

6.194.2.1 TypeConsistencyEnforcementKind

`typedef` **`dds::core::safe_enum< TypeConsistencyEnforcementKind_def> dds::core::policy::Type↩ ConsistencyEnforcementKind`**

Safe Enumeration (p. 429) of **TypeConsistencyEnforcementKind_def** (p. 2570)

See also

TypeConsistencyEnforcementKind_def (p. 2570)

6.195 TYPESUPPORT

[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Allows you to attach application-specific values to a **dds::pub::↩ DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.

Classes

- class **rti::core::policy::TypeSupport**

  <<***extension***>> *(p. 116) Allows attaching application-specific information to a **dds::pub::DataWriter** (p. 1138) or **dds::↩ sub::DataReader** (p. 976) that is passed to the serilization and deserialization routines.*
- struct **rti::core::policy::CdrPaddingKind_def**

  <<***extension***>> *(p. 354) The definition of the **dds::core::safe_enum** (p. 2265) CdrPaddingKind*

Typedefs

- typedef **dds::core::safe_enum< CdrPaddingKind_def > rti::core::policy::CdrPaddingKind**

  <<***extension***>> *(p. 354) **Safe Enumeration** (p. 429) of **CdrPaddingKind_def** (p. 920)*

6.195.1 Detailed Description

[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Allows you to attach application-specific values to a **dds::pub::↩ DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.

6.195.2 Typedef Documentation

6.195.2.1 CdrPaddingKind

```
typedef dds::core::safe_enum< CdrPaddingKind_def> rti::core::policy::CdrPaddingKind
```

<<***extension***>> *(*p. *354)* **Safe Enumeration** (p. 429) of **CdrPaddingKind_def** (p. 920)

See also

CdrPaddingKind_def (p. 920)

6.196 USER_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 46) during discovery.

Classes

- class **dds::core::policy::UserData**
  *Attaches a buffer of opaque data that is distributed by **Built-in Topics** (*p. *46) during discovery.*

6.196.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 46) during discovery.

6.197 WRITER_DATA_LIFECYCLE

<<***spec***>> *(*p. *112)* Controls how a DataWriter handles the lifecycle of the instances (keys) that it is registered to manage.

Classes

- class **dds::core::policy::WriterDataLifecycle**
  *Controls how a **dds::pub::DataWriter** (*p. *1138) handles the lifecycle of the instances (keys) that it writes.*

6.197.1 Detailed Description

<<***spec***>> *(*p. *112)* Controls how a DataWriter handles the lifecycle of the instances (keys) that it is registered to manage.

6.198 WIRE_PROTOCOL

<<***extension***>> *(*p. *116)* Specifies the wire protocol related attributes for the **dds::domain::DomainParticipant** (p. 1326).

Classes

- class **rti::core::policy::WireProtocol**

  <<***extension***>> *(*p. *116) Configures the write protocol of a* ***dds::domain::DomainParticipant*** *(*p. *1326)*
- struct **rti::core::policy::WireProtocolAutoKind_def**

  <<***extension***>> *(*p. *354) The definition of the* ***dds::core::safe_enum*** *(*p. *2265) WireProtocolAutoKind*
- class **rti::core::policy::RtpsReservedPortKindMask**

  <<***extension***>> *(*p. *354) Mask of reserved ports*
- class **rti::core::RtpsWellKnownPorts**

  <<***extension***>> *(*p. *354) Configures the mapping of the RTPS well-known ports*

Typedefs

- typedef **dds::core::safe_enum**< **WireProtocolAutoKind_def** > **rti::core::policy::WireProtocolAutoKind**

  <<***extension***>> *(*p. *354)* ***Safe Enumeration*** *(*p. *429) of* ***WireProtocolAutoKind_def*** *(*p. *2641)*

Functions

- static const **RtpsReservedPortKindMask** **rti::core::policy::RtpsReservedPortKindMask::all** ()

  All bits are set.
- static const **RtpsReservedPortKindMask** **rti::core::policy::RtpsReservedPortKindMask::none** ()

  No bits are set.
- static const **RtpsReservedPortKindMask** **rti::core::policy::RtpsReservedPortKindMask::default_mask** ()

  The default value of ***rti::core::policy::WireProtocol::rtps_reserved_port_mask*** *(*p. *2639).*
- static **RtpsWellKnownPorts** **rti::core::RtpsWellKnownPorts::Interoperable** ()

  Returns an instance containing the port mapping compliant with the OMG DDS Interoperability wire protocol.
- static **RtpsWellKnownPorts** **rti::core::RtpsWellKnownPorts::BackwardsCompatible** ()

  Returns an instance containing the port mapping compatible with previous versions of RTI Connext.

6.198.1 Detailed Description

<<***extension***>> *(*p. *116)* Specifies the wire protocol related attributes for the **dds::domain::DomainParticipant** (p. 1326).

6.198.2 Typedef Documentation

6.198.2.1 WireProtocolAutoKind

```
typedef dds::core::safe_enum< WireProtocolAutoKind_def> rti::core::policy::WireProtocolAutoKind
```

<<***extension***>> *(*p. *354)* **Safe Enumeration** (p. 429) of **WireProtocolAutoKind_def** (p. 2641)

See also

WireProtocolAutoKind_def (p. 2641)

6.198.3 Function Documentation

6.198.3.1 all()

```
static const RtpsReservedPortKindMask rti::core::policy::RtpsReservedPortKindMask::all ( ) [inline],
[static]
```

All bits are set.

All of the ports that may be needed by DDS will be reserved when the participant is enabled. With this value set, failure to allocate a port that is computed based on the **rti::core::RtpsWellKnownPorts** (p. 2258) will be detected at this time, and the enable operation will fail.

Note that this will also reserve the **usertraffic** multicast port which is not actually used unless there are DataReaders that enable multicast but fail to specify a port. To avoid unnecesary resource usage for these ports, use RTPS_↩ RESERVED_PORT_MASK_DEFAULT.

Automatic participant ID selection will be based on finding a participant index with both the discovery (metatraffic) unicast port and usertraffic unicast port available.

See also

RtpsReservedPortKindMask (p. 2255)

References **rti::core::policy::RtpsReservedPortKindMask::RtpsReservedPortKindMask()**.

6.198.3.2 none()

```
static const RtpsReservedPortKindMask rti::core::policy::RtpsReservedPortKindMask::none ( ) [inline],
[static]
```

No bits are set.

None of the ports that are needed by DDS will be allocated until they are specifically required. With this value set, automatic participant Id selection will be based on selecting a port for discovery (metatraffic) unicast traffic on a single transport.

See also

RtpsReservedPortKindMask (p. 2255)

References **rti::core::policy::RtpsReservedPortKindMask::RtpsReservedPortKindMask()**.

6.198.3.3 default_mask()

```
static const RtpsReservedPortKindMask rti::core::policy::RtpsReservedPortKindMask::default_mask (
) [inline], [static]
```

The default value of **rti::core::policy::WireProtocol::rtps_reserved_port_mask** (p. 2639).

Most of the ports that may be needed by DDS will be reserved by the transport when the participant is enabled. With this value set, failure to allocate a port that is computed based on the **rti::core::RtpsWellKnownPorts** (p. 2258) will be detected at this time and the enable operation will fail.

This setting will avoid reserving the **usertraffic** multicast port, which is not actually used unless there are DataReaders that enable multicast but fail to specify a port.

Automatic participant ID selection will be based on finding a participant index with both the discovery (metatraffic) unicast port and usertraffic unicast port available.

See also

RtpsReservedPortKindMask (p. 2255)

References **rti::core::policy::RtpsReservedPortKindMask::RtpsReservedPortKindMask()**.

6.198.3.4 Interoperable()

```
static RtpsWellKnownPorts rti::core::RtpsWellKnownPorts::Interoperable ( ) [inline], [static]
```

Returns an instance containing the port mapping compliant with the OMG DDS Interoperability wire protocol.

Assign **rti::core::policy::WireProtocol::rtps_well_known_ports** (p. 2638) to this value to use well-known port mappings which are compliant with OMG's DDS Interoperability Wire Protocol.

The following are the `rtps_well_known_ports` values for **rti::core::RtpsWellKnownPorts::Interoperable()** (p. 557):

`port_base` = 7400
`domain_id_gain` = 250
`participant_id_gain` = 2
`builtin_multicast_port_offset` = 0
`builtin_unicast_port_offset` = 10
`user_multicast_port_offset` = 1
`user_unicast_port_offset` = 11

Assuming a maximum port number of 65535 (UDPv4), the above settings enable the use of about 230 domains with up to 120 Participants per node per domain.

These settings are *not* backwards compatible with previous versions of the RTI Connext middleware that used fixed port mappings. For backwards compability, please use **rti::core::RtpsWellKnownPorts::BackwardsCompatible()** (p. 558).

See also

rti::core::policy::WireProtocol::rtps_well_known_ports (p. 2638)

rti::core::RtpsWellKnownPorts::BackwardsCompatible() (p. 558)

6.198.3.5 BackwardsCompatible()

```
static RtpsWellKnownPorts rti::core::RtpsWellKnownPorts::BackwardsCompatible ( ) [inline], [static]
```

Returns an instance containing the port mapping compatible with previous versions of RTI Connext.

Assign **rti::core::policy::WireProtocol::rtps_well_known_ports** (p. 2638) to this value to remain compatible with previous versions of the RTI Connext middleware that used fixed port mappings.

The following are the `rtps_well_known_ports` values for **rti::core::RtpsWellKnownPorts::Backwards↩ Compatible()** (p. 558):

`port_base` = 7400
`domain_id_gain` = 10
`participant_id_gain` = 1000
`builtin_multicast_port_offset` = 2
`builtin_unicast_port_offset` = 0
`user_multicast_port_offset` = 1
`user_unicast_port_offset` = 3

These settings are *not* compliant with OMG's DDS Interoperability Wire Protocol. To comply with the specification, please use **rti::core::RtpsWellKnownPorts::Interoperable()** (p. 557).

See also

rti::core::policy::WireProtocol::rtps_well_known_ports (p. 2638)

rti::core::RtpsWellKnownPorts::Interoperable() (p. 557)

6.199 NDDS_DISCOVERY_PEERS

Environment variable or a file that specifies the default values of **rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers)** (p. 1268) and **rti::core::policy::Discovery::multicast_receive_addresses** (p. 1268) contained in the dds::domain::qos::DomainParticipantQos::discovery qos policy.

Environment variable or a file that specifies the default values of **rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers)** (p. 1268) and **rti::core::policy::Discovery::multicast_receive_addresses** (p. 1268) contained in the dds::domain::qos::DomainParticipantQos::discovery qos policy.

The default value of the **dds::domain::qos::DomainParticipantQos** (p. 1384) is obtained by calling **dds::domain::DomainParticipant::default_participant_qos()** (p. 1341)().

NDDS_DISCOVERY_PEERS specifies the default value of the **rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers)** (p. 1268) and **rti::core::policy::Discovery::multicast_receive_addresses** (p. 1268) fields, when the default participant QoS policies have not been explictly set by the user (i.e., **dds::domain::DomainParticipant::default_participant_qos()** (p. 1341)() has never been called or was called using PARTICIPANT_QOS_DEFAULT).

If NDDS_DISCOVERY_PEERS does *not* contain a multicast address, then the string sequence **rti::core::policy::Discovery::multicast_receive_addresses** (p. 1268) is cleared and the RTI discovery process will not listen for discovery messages via multicast.

If NDDS_DISCOVERY_PEERS contains one or more multicast addresses, the addresses will be stored in **rti::core::policy::Discovery::multicast_receive_addresses** (p. 1268), starting at element 0. They will be stored in the order in which they appear in NDDS_DISCOVERY_PEERS.

Note: IPv4 multicast addresses must have a prefix. Therefore, when using the UDPv6 transport: if there are any IPv4 multicast addresses in the peers list, make sure they have "udpv4://" in front of them (such as udpv4://239.255.0.1).

Note: Currently, RTI Connext will only listen for discovery traffic on the first multicast address (element 0) in **rti::core::policy::Discovery::multicast_receive_addresses** (p. 1268).

NDDS_DISCOVERY_PEERS provides a mechanism to dynamically switch the discovery configuration of an RTI Connext application without recompilation. The application programmer is free to not use the default values; instead use values supplied by other means.

NDDS_DISCOVERY_PEERS can be specified either in an environment variable as comma (',') separated "peer descriptors" (see **Peer Descriptor Format** (p. 560)) or in a file. These formats are described below.

6.199.1 Peer Descriptor Format

A **peer descriptor** string specifies a range of participants at a given `locator`. Peer descriptor strings are used in the **rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers)** (p. 1268) field and the **dds::domain::DomainParticipant::add_peer(const std::string & peer_descr_string)** (p. 1352)() operation.

The anatomy of a `peer` descriptor is illustrated below using a UDPv4 transport and a custom "StarFabric" transport example.

A peer descriptor consists of:

optional **Participant ID Limit**. If a simple integer is specified, it indicates the maximum participant ID to be contacted by the RTI Connext discovery mechanism at the given locator. If that integer is enclosed in square brackets (e.g.: [2]) *only* that Participant ID will be used. You can also specify a range in the form of [a-b]: in this case only the Participant IDs in that specific range are contacted. If omitted, a default value of 4 is implied: participant IDs 0,1,2,3, and 4 will be contacted.

- **Locator (**p. **1673)**. See **Locator Format** (p. 560).

These are separated by the '@' character. The separator may be omitted if a participant ID limit is not explictly specified.

Note that the "participant ID limit" only applies to unicast locators; it is ignored for multicast locators (and therefore should be omitted for multicast peer descriptors).

6.199.1.1 Locator Format

A **locator** string specifies a transport and an address in string format. Locators are used to form peer descriptors. A locator is equivalent to a peer descriptor with the default maximum participant ID.

A locator consists of:

optional **Transport name** (**alias** or class). This identifies the set of transport plugins (**Transport Aliases** (p. 76)) that may be used to parse the `address` portion of the locator. Note that a transport class name is an implicit alias that is used to refer to all the transport plugin instances of that class.

optional **Address**. See **Address Format** (p. 561).

These are separated by the "//:" string. The separator is specified if and only if a transport name is specified.

If a transport name is specified, the address may be omitted; in that case, all the unicast addresses (across all transport plugin instances) associated with the transport class are implied. Thus, a locator string may specify several addresses.

If an address is specified, the transport name and the separator string may be omitted; in that case all the available transport plugins (for the **dds::core::Entity** (p. 1520)) may be used to parse the address string.

6.199.1.2 Address Format

An **address** string specifies a transport-independent network address that qualifies a **transport-dependent** address string. Addresses are used to form locators. Addresses are also used in **rti::core::policy::Discovery::multicast_↩ receive_addresses** (p. 1268), and **rti::core::TransportMulticastSettings::receive_address** (p. 2551) fields. An address is equivalent to a locator in which the transport name and separator are omitted.

An address consists of:

optional **Network Address**. An address in IPv4 or IPv6 string notation. If omitted, the network address of the transport is implied (**Transport Network Address** (p. 79)).

optional **Transport Address**. A string that is passed to the transport for processing. The transport maps this string into **NDDS_Transport_Property_t::address_bit_count** (p. 1788) bits. If omitted the network address is used as the fully qualified address.

These are separated by the '#' character. If a separator is specified, it must be followed by a non-empty string which is passed to the transport plugin. If the separator is omitted, it is treated as a transport address with an implicit network address (of the transport plugin). The implicit network address is the address used when registering the transport: e.g, the UDPv4 implicit network address is 0.0.0.0.0.0.0.0.0.0.0.0.

The bits resulting from the transport address string are combined with the network address. The least or most significant **NDDS_Transport_Property_t::address_bit_count** (p. 1788) bits of the network address are ignored and replaced with the transport address. The sign of the **NDDS_Transport_Property_t::address_bit_count** (p. 1788) value determines whether the most significant bits (negative sign) or least significant bits (positive sign) are ignored (**Transport Network Address** (p. 79)).

You can use a hostname instead of a transport address. If a hostname is used, the transport will attempt to resolve the hostname upon enabling the DomainParticipant. Changes to the hostname resolution will only be detected if the DNS tracker is enabled (it is enabled by default); see `Using DNS Tracker to Keep Peer List Updated`, in the Core Libraries User's Manual. For example:

```
<domain_participant_qos>

    <discovery>

        <initial_peers>

            <element>udpv4://myhostname</element>

        </initial_peers>

    </discovery>

</domain_participant_qos>
```

In this example, the DomainParticipant will attempt to resolve "myhostname" using the UDPv4 transport.

6.199.2 NDDS_DISCOVERY_PEERS Environment Variable Format

NDDS_DISCOVERY_PEERS can be specified via an environment variable of the same name, consisting of a sequence of peer descriptors separated by the comma (',') character.

Examples

Multicast (maximum participant ID is irrelevant)

- 239.255.0.1

Default maximum participant ID on localhost

- localhost

Default maximum participant ID on host 192.168.1.1 (IPv4)

- 192.168.1.1

Default maximum participant ID on host FAA0::0 (IPv6)

- FAA0::1

Default maximum participant ID on host himalaya accessed using the "udpv4" transport plugin(s) (IPv4)

- udpv4://himalaya

Default maximum participant ID on localhost using the "udpv4" transport plugin(s) registered at network address FAA0::0

- udpv4://FAA0::0#localhost

Default maximum participant ID on host 0/0/R (StarFabric)

- 0/0/R
- #0/0/R

Default maximum participant ID on host 0/0/R (StarFabric) using the "starfabric" (StarFabric) transport plugin(s)

- starfabric://0/0/R
- starfabric://#0/0/R

Default maximum participant ID on host 0/0/R (StarFabric) using the "starfabric" (StarFabric) transport plugin(s) registered at network address FAA0::0

- starfabric://FBB0::0#0/0/R

Default maximum participant ID on all unicast addresses accessed via the "starfabric" (StarFabric) transport plugin(s)

- starfabric://

Default maximum participant ID on all unicast addresses accessed via the "shmem" (shared memory) transport plugin(s) registered at network address FCC0::0

- shmem://FCC0::0

Default maximum participant ID on hosts himalaya and gangotri

- himalaya,gangotri

Maximum participant ID of 1 on hosts himalaya and gangotri

- 1@himalaya,1@gangotri

Combinations of above

- 239.255.0.1,localhost,192.168.1.1,0/0/R
- FAA0::1,FAA0::0#localhost,FBB0::0#0/0/R
- udpv4://himalaya,udpv4://FAA0::0#localhost,#0/0/R
- starfabric://0/0/R,starfabric://FBB0::0#0/0/R,shmem://
- starfabric://,shmem://FCC0::0,1@himalaya,1@gangotri

6.199.3 NDDS_DISCOVERY_PEERS File Format

NDDS_DISCOVERY_PEERS can be specified via a file of the same name in the program's current working directory. A NDDS_DISCOVERY_PEERS file would contain a sequence of peer descriptors separated by whitespace or the comma (',') character. The file may also contain comments starting with a semicolon (';') character till the end of the line.

Example:

```
;; NDDS_DISCOVERY_PEERS - Discovery Configuration File
;;
;;
;; NOTE:
;;   1. This file must be in the current working directory, i.e.
;;      in the folder from which the application is launched.
;;
;;   2. This file takes precedence over the environment variable NDDS_DISCOVERY_PEERS
;;
;; Multicast
```

```
239.255.0.1                 ; The default dds discovery multicast address
;; Unicast
localhost,192.168.1.1       ; A comma can be used a separator
FAA0::1  FAA0::0#localhost  ; Whitespace can be used as a separator
1@himalaya                  ; Maximum participant ID of 1 on 'himalaya'
1@gangotri
;; UDPv4
udpv4://himalaya            ; 'himalaya' via 'udpv4' transport plugin(s)
udpv4://FAA0::0#localhost   ; 'localhost' via 'updv4' transport
                            ;  plugin registered at network address FAA0::0
;; Shared Memory
shmem://                    ; All 'shmem' transport plugin(s)
builtin.shmem://            ; The builtin 'shmem' transport plugin
shmem://FCC0::0             ; Shared memory transport plugin registered
                            ;                at network address FCC0::0
;; StarFabric
0/0/R                       ; StarFabric node 0/0/R
starfabric://0/0/R          ; 0/0/R accessed via 'starfabric'
                            ;             transport plugin(s)
starfabric://FBB0::0#0/0/R  ; StarFabric transport plugin registered
                            ;             at network address FBB0::0
starfabric://               ; All 'starfabric' transport plugin(s)
```

6.199.4 NDDS_DISCOVERY_PEERS Precedence

If the current working directory from which the RTI Connext application is launched contains a file called NDDS_↩ DISCOVERY_PEERS, and an environment variable named NDDS_DISCOVERY_PEERS is also defined, the file takes precedence; the environment variable is ignored.

6.199.5 NDDS_DISCOVERY_PEERS Default Value

If NDDS_DISCOVERY_PEERS is not specified (either as a file in the current working directory, or as an environment variable), it implicitly defaults to the following.

```
;; Multicast (only on platforms which allow UDPv4 multicast out of the box)
;;
;; This allows any dds applications anywhere on the local network to
```

```
;; discover each other over UDPv4.
builtin.udpv4://239.255.0.1  ; dds's default discovery multicast address
                             ; This is also the default multicast receive address
;; Unicast - UDPv4 (on all platforms)
;;
;; This allows two dds applications using participant IDs up to the maximum
;; default participant ID on the local host and domain to discover each
;; other over UDP/IPv4.
builtin.udpv4://127.0.0.1
;; Unicast - Shared Memory (only on platforms that support shared memory)
;;
;; This allows two dds applications using participant IDs up to the maximum
;; default participant ID on the local host and domain to discover each
;; other over shared memory.
builtin.shmem://
```

6.199.6 Builtin Transport Class Names

The class names for the builtin transport plugins are:

- `shmem` - ShmemTransport
- `udpv4` - UDPv4Transport
- `udpv6` - UDPv6Transport

These may be used as the transport names in the **Locator Format** (p. 560).

6.199.7 NDDS_DISCOVERY_PEERS and Local Host Communication

Suppose you want to communicate with other RTI Connext applications on the same host and you are setting NDDS↩ _DISCOVERY_PEERS explicitly (generally in order to use unicast discovery with applications on other hosts).

If the local host platform does not support the shared memory transport, then you can include the name of the local host in the NDDS_DISCOVERY_PEERS list.

If the local host platform supports the shared memory transport, then you can do one of the following:

- Include "shmem://" in the NDDS_DISCOVERY_PEERS list. This will cause shared memory to be used for discovery and data traffic for applications on the same host.

or:

- Include the name of the local host in the NDDS_DISCOVERY_PEERS list and disable the shared memory transport in the **rti::core::policy::TransportBuiltin** (p. 2533) of the **dds::domain::DomainParticipant** (p. 1326). This will cause UDP loopback to be used for discovery and data traffic for applications on the same host.

(To check if your platform supports shared memory, see the `Platform Notes`.)

See also

rti::core::policy::Discovery::multicast_receive_addresses (p. 1268)

rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers) (p. 1268)

dds::domain::DomainParticipant::add_peer(const std::string & peer_descr_string) (p. 1352)()

PARTICIPANT_QOS_DEFAULT

dds::domain::DomainParticipant::default_participant_qos() (p. 1341)()

Transport Aliases (p. 76)

Transport Network Address (p. 79)

6.200 SampleProcessor

*<<**experimental**>> (p. 355) <<**extension**>> (p. 354)* Utility to concurrently read and process the data samples received by **dds::sub::DataReader** (p. 976).

Classes

- class **rti::sub::SampleProcessor**

  *<<**extension**>> (p. 354) <<**reference-type**>> (p. 351) Utility to read and process the data samples that one or more DataReaders receive using a sample handler.*

6.200.1 Detailed Description

*<<**experimental**>> (p. 355) <<**extension**>> (p. 354)* Utility to concurrently read and process the data samples received by **dds::sub::DataReader** (p. 976).

6.201 ServiceRequest Built-in Topic

Builtin topic for accessing requests from different services within RTI Connext.

Classes

- struct **rti::core::ServiceRequestId_def**

  The definition of the rti::core::safe_enum ServiceRequestId.
- class **rti::topic::ServiceRequest**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A request coming from one of the built-in services*

Typedefs

- typedef **dds::core::safe_enum< ServiceRequestId_def > rti::core::ServiceRequestId**

  ***Safe Enumeration** (p. 429) of **ServiceRequestId_def** (p. 2361)*

Functions

- std::string **rti::topic::service_request_topic_name** ()

  ***ServiceRequest** (p. 2358) built-in topic name.*

6.201.1 Detailed Description

Builtin topic for accessing requests from different services within RTI Connext.

Currently, the **rti::sub::TopicQuery** (p. 2516), Locator Reachability Instance State Consistency and Controlability (part of Observability) all rely on this topic.

See also

Topic Queries (p. 67) for an explanation of how TopicQueries use ServiceRequests and how you can access the ServiceRequests for debugging purposes in the section **The Built-in ServiceRequest DataReader** (p. 70).

6.201.2 Typedef Documentation

6.201.2.1 ServiceRequestId

```
typedef dds::core::safe_enum< ServiceRequestId_def> rti::core::ServiceRequestId
```

Safe Enumeration (p. 429) of **ServiceRequestId_def** (p. 2361)

See also

ServiceRequestId_def (p. 2361)

6.201.3 Function Documentation

6.201.3.1 service_request_topic_name()

```
std::string rti::topic::service_request_topic_name ( )
```

ServiceRequest (p. 2358) built-in topic name.

Topic name of the builtin **dds::sub::DataReader** (p. 976) for the **rti::topic::ServiceRequest** (p. 2358) type

6.202 Topic-type serialization and deserialization

Provides functions to serialize and deserialize user data types to and from Common Data Representation (CDR) format.

Functions

- template<typename TopicType >
  void **rti::topic::from_cdr_buffer_no_alloc** (TopicType &sample, const std::vector< char > &buffer)

  Deserializes a sample from a buffer of bytes in CDR format.
- template<typename TopicType >
  TopicType **rti::topic::from_cdr_buffer** (const std::vector< char > &buffer)

  Creates a sample by deserializing a buffer of bytes in CDR format.
- template<typename TopicType >
  std::vector< char > & **rti::topic::to_cdr_buffer** (std::vector< char > &buffer, const TopicType &sample, **dds::core::policy::DataRepresentationId** representation)

  Serializes a sample into a buffer of octets in CDR format.
- template<typename TopicType >
  std::vector< char > & **rti::topic::to_cdr_buffer** (std::vector< char > &buffer, const TopicType &sample)

  Serializes a sample into a buffer of octets in CDR format.

6.202.1 Detailed Description

Provides functions to serialize and deserialize user data types to and from Common Data Representation (CDR) format.

CDR defines how a data sample is encoded before being sent over the network. In DDS, topic-types are serialized into bytes in CDR format to transmit them between DataWriters and DataReaders. The functions rti::topic::to_cdr() and rti::topic::from_cdr() expose this functionality so applications can take advantage of this capability. For more information, see `Data Representation` in the Core Libraries Extensible Types Guide.

6.202.2 Function Documentation

6.202.2.1 from_cdr_buffer_no_alloc()

```
template<typename TopicType >
void rti::topic::from_cdr_buffer_no_alloc (
            TopicType & sample,
            const std::vector< char > & buffer )
```

Deserializes a sample from a buffer of bytes in CDR format.

This function does the same as **from_cdr_buffer()** (p. 569) but it doesn't create a new sample. When deserializing multiple samples this operation will be more efficient.

Template Parameters

TopicType	A valid topic-type. Valid types are **IDL-generated types** (p. 606), the **built-in types** (p. 50) and **dds::core::xtypes::DynamicData** (p. 1461).

Parameters

sample	The destination (see precondition)
buffer	The CDR buffer to deserialize

References **rti::topic::from_cdr_buffer()**.

Referenced by **rti::topic::from_cdr_buffer()**.

6.202.2.2 from_cdr_buffer()

```
template<typename TopicType >
TopicType rti::topic::from_cdr_buffer (
            const std::vector< char > & buffer )
```

Creates a sample by deserializing a buffer of bytes in CDR format.

Template Parameters

TopicType	A valid topic-type. Valid types are **IDL-generated types** (p. 606), the **built-in types** (p. 50) and **dds::core::xtypes::DynamicData** (p. 1461).

Parameters

buffer	The CDR buffer to desrialize

Returns

The deserialized sample

See also

from_cdr_buffer_no_alloc() (p. 568)

References **rti::topic::from_cdr_buffer_no_alloc()**.

Referenced by **rti::topic::from_cdr_buffer_no_alloc()**.

6.202.2.3 to_cdr_buffer() [1/2]

```
template<typename TopicType >
std::vector< char > & rti::topic::to_cdr_buffer (
            std::vector< char > & buffer,
            const TopicType & sample,
             dds::core::policy::DataRepresentationId representation )
```

Serializes a sample into a buffer of octets in CDR format.

Template Parameters

TopicType	A valid topic-type. Valid types are **IDL-generated types** (p. 606), the **built-in types** (p. 50) and **dds::core::xtypes::DynamicData** (p. 1461).

Parameters

buffer	The destination buffer. It will be resized to fit the serialized sample
sample	The sample to serialize
representation	The representation to serialize the data into

Returns

A reference to the parameter `buffer`

References **rti::topic::to_cdr_buffer()**.

6.202.2.4 to_cdr_buffer() [2/2]

```
template<typename TopicType >
std::vector< char > & rti::topic::to_cdr_buffer (
```

```
std::vector< char > & buffer,
const TopicType & sample )
```

Serializes a sample into a buffer of octets in CDR format.

This overload uses dds::core::policy::DataRepresentationId::auto_id() as the data representation.

References **rti::topic::to_cdr_buffer()**.

Referenced by **rti::topic::to_cdr_buffer()**.

6.203 Custom Content Filters

Classes and associated types used to implement custom content filters.

Classes

- struct **rti::topic::no_compile_data_t**

  *The type to specify as the CompileData template parameter to your **ContentFilter** (p. 950) if your compile function does not return any data.*
- class **rti::topic::ContentFilter< T, CompileData >**

  *<<**extension**>> (p. 354) A class to inherit from when implementing a custom content filter*
- class **rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >**

  *<<**extension**>> (p. 354) A class to inherit from when implementing a writer-side custom content filter*
- class **rti::topic::WriterContentFilterHelper< T, CompileData, WriterFilterData >**

  *<<**extension**>> (p. 354) A class to inherit from when implementing a writer-side custom content filter.*
- class **rti::topic::CustomFilter< T >**

  *<<**extension**>> (p. 354) <<**reference-type**>> (p. 351) A wrapper class for the user-defined implementation of a **ContentFilter** (p. 950).*
- class **rti::topic::ExpressionProperty**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Provides additional information about the filter expression passed to the writer_compile method of **rti::topic::WriterContentFilter** (p. 2652)*
- class **rti::topic::FilterSampleInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Provides meta information associated with the sample.*

Functions

- template<typename T >
  CustomFilter< T > rti::topic::find_content_filter (const **dds::domain::DomainParticipant** &participant, const std::string &filter_name)

  *Lookup a content filter previously registered with **dds::domain::DomainParticipant::register_contentfilter** (p. 1349).*

Variables

- **no_compile_data_t no_compile_data**

  A constant to return if your compile function does not create any compile data

6.203.1 Detailed Description

Classes and associated types used to implement custom content filters.

6.203.2 Function Documentation

6.203.2.1 find_content_filter()

```
template<typename T >
CustomFilter< T > rti::topic::find_content_filter (
            const  dds::domain::DomainParticipant & participant,
            const std::string & filter_name )
```

Lookup a content filter previously registered with **dds::domain::DomainParticipant::register_contentfilter** (p. 1349).

```
#include <dds/topic/find.hpp>
```

Note

This is a standalone function in the namespace **rti::topic** (p. 771)

You cannot lookup the RTI Connext built-in content filters.

Template Parameters

T	The user-defined content filter type that the **CustomFilter** (p. 967) was created with

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) that the content filter is registered with.
filter_name	Name of the filter to lookup.

Returns

The found **rti::topic::CustomFilter** (p. 967) or **dds::core::null** (p. 438) if it is not found

References **dds::core::null**.

6.203.3 Variable Documentation

6.203.3.1 no_compile_data

```
template<typename T , typename CompileData = no_compile_data_t>
no_compile_data_t no_compile_data  [related]
```

A constant to return if your compile function does not create any compile data

6.204 Discovery Snapshot

Utilities to print discovery snapshots of DDS entities.

Functions

- void **rti::util::discovery::take_snapshot** (**dds::domain::DomainParticipant** participant)

  <<***extension***>> *(*p. *354) Take a snapshot of the remote participants discovered by a local one.*
- void **rti::util::discovery::take_snapshot** (**dds::domain::DomainParticipant** participant, const std::string &file_name)

  <<***extension***>> *(*p. *354) Take a snapshot of the remote participants discovered by a local one.*
- void **rti::util::discovery::take_snapshot** (**dds::pub::AnyDataWriter** writer)

  <<***extension***>> *(*p. *354) Take a snapshot of the compatible and incompatible remote readers matched by a local writer.*
- void **rti::util::discovery::take_snapshot** (**dds::pub::AnyDataWriter** writer, const std::string &file_name)

  <<***extension***>> *(*p. *354) Take a snapshot of the compatible and incompatible remote readers matched by a local writer.*
- void **rti::util::discovery::take_snapshot** (**dds::sub::AnyDataReader** reader)

  <<***extension***>> *(*p. *354) Take a snapshot of the compatible and incompatible remote writers matched by a local reader.*
- void **rti::util::discovery::take_snapshot** (**dds::sub::AnyDataReader** reader, const std::string &file_name)

  <<***extension***>> *(*p. *354) Take a snapshot of the compatible and incompatible remote writers matched by a local reader.*

6.204.1 Detailed Description

Utilities to print discovery snapshots of DDS entities.

6.204.2 Function Documentation

6.204.2.1 take_snapshot() [1/6]

```
void rti::util::discovery::take_snapshot (
              dds::domain::DomainParticipant participant )
```

<<***extension***>> *(*p. *354)* Take a snapshot of the remote participants discovered by a local one.

Note

This is a standalone function in the namespace **rti::util::discovery** (p. 778)

Parameters

participant	The local participant.

The snapshot will be printed through the **rti::config::Logger** (p. 1683). A possible output may be the following:

```
Remote participants that match the local participant domain=0

name="participantTestName" role="participantTestRole" id="1"

guid_prefix="0x0101D8D1,0x20B83C0D,0x4495246E"

------------------------------------------------------------------------

1. 0x0101542A,0x2C59B595,0xA1693BDF name="participantTestName"

role="participantTestRole"

unicastLocators="udpv4://192.168.1.170:7411"

------------------------------------------------------------------------
```

Exceptions

One	of the **Standard Exceptions** (p. 428).

6.204.2.2 take_snapshot() [2/6]

```
void rti::util::discovery::take_snapshot (
              dds::domain::DomainParticipant participant,
             const std::string & file_name )
```

<<***extension***>> *(p. 354)* Take a snapshot of the remote participants discovered by a local one.

Note

This is a standalone function in the namespace **rti::util::discovery** (p. 778)

Parameters

participant	The local participant.
file_name	Name of the file where snapshot should be printed.

The snapshot will be printed in the file specified by file_name. A possible output may be the following:

```
Remote participants that match the local participant domain=0

name="participantTestName" role="participantTestRole" id="1"

guid_prefix="0x0101D8D1,0x20B83C0D,0x4495246E"

------------------------------------------------------------------------

1. 0x0101542A,0x2C59B595,0xA1693BDF name="participantTestName"
```

```
role="participantTestRole"

unicastLocators="udpv4://192.168.1.170:7411"

----------------------------------------------------------------------
```

Exceptions

One	of the **Standard Exceptions** (p. 428).

6.204.2.3 take_snapshot() [3/6]

```
void rti::util::discovery::take_snapshot (
              dds::pub::AnyDataWriter writer )
```

<<***extension***>> *(p. 354)* Take a snapshot of the compatible and incompatible remote readers matched by a local writer.

Note

This is a standalone function in the namespace **rti::util::discovery** (p. 778)

Parameters

writer	The local writer.

The snapshot will be printed through the **rti::config::Logger** (p. 1683). A possible output may be the following:

```
Remote readers that match the local writer domain=0 name="writerTestName"

guid="0x0101D8D1,0x20B83C0D,0x4495246E:0x80000003"

topic="FooTopic" type="FooType"

----------------------------------------------------------------------------

Compatible readers:

1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000004 name="reader1TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

liveliness="ALIVE"

Incompatible readers:

1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000104 name="reader2TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

reason="Inconsistent QoS"

----------------------------------------------------------------------------
```

Exceptions

One	of the **Standard Exceptions** (p. 428).

6.204.2.4 take_snapshot() [4/6]

```
void rti::util::discovery::take_snapshot (
              dds::pub::AnyDataWriter writer,
             const std::string & file_name )
```

<<***extension***>> *(p. 354)* Take a snapshot of the compatible and incompatible remote readers matched by a local writer.

Note

This is a standalone function in the namespace **rti::util::discovery** (p. 778)

Parameters

writer	The local writer.
file_name	Name of the file where snapshot should be printed.

The snapshot will be printed in the file specified by file_name. A possible output may be the following:

```
Remote readers that match the local writer domain=0 name="writerTestName"

guid="0x0101D8D1,0x20B83C0D,0x4495246E:0x80000003"

topic="FooTopic" type="FooType"

------------------------------------------------------------------------

Compatible readers:

1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000004 name="reader1TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

liveliness="ALIVE"

Incompatible readers:

1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000104 name="reader2TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

reason="Inconsistent QoS"

------------------------------------------------------------------------
```

Exceptions

One	of the **Standard Exceptions** (p. 428).

6.204.2.5 take_snapshot() [5/6]

```
void rti::util::discovery::take_snapshot (
            dds::sub::AnyDataReader reader )
```

<<***extension***>> *(p. 354)* Take a snapshot of the compatible and incompatible remote writers matched by a local reader.

Note

This is a standalone function in the namespace **rti::util::discovery** (p. 778)

Parameters

reader	Local reader.

The snapshot will be printed through the **rti::config::Logger** (p. 1683). A possible output may be the following:

```
Remote writers that match the local reader domain=0 name="readerTestName"

guid="0x0101542A,0x2C59B595,0xA1693BDF:0x80000004"

topic="FooTopic" type="FooType"

------------------------------------------------------------------------

Compatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000003 name="writer1TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

liveliness="ALIVE"

Incompatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000103 name="writer2TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

reason="Inconsistent QoS"

------------------------------------------------------------------------
```

Exceptions

One	of the **Standard Exceptions** (p. 428).

6.204.2.6 take_snapshot() [6/6]

```
void rti::util::discovery::take_snapshot (
```

```
dds::sub::AnyDataReader reader,
const std::string & file_name )
```

<<***extension***>> *(p. 354)* Take a snapshot of the compatible and incompatible remote writers matched by a local reader.

Note

This is a standalone function in the namespace **rti::util::discovery** (p. 778)

Parameters

reader	Local reader.
file_name	Name of the file where snapshot should be printed.

The snapshot will be printed in the file specified by file_name. A possible output may be the following:

```
Remote writers that match the local reader domain=0 name="readerTestName"

guid="0x0101542A,0x2C59B595,0xA1693BDF:0x80000004"

topic="FooTopic" type="FooType"

------------------------------------------------------------------------

Compatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000003 name="writer1TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

liveliness="ALIVE"

Incompatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000103 name="writer2TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

reason="Inconsistent QoS"

------------------------------------------------------------------------
```

Exceptions

One	of the **Standard Exceptions** (p. 428).

6.205 Network Capture

Save network traffic into a capture file for further analysis.

Classes

- class **rti::util::network_capture::ContentKindMask**

*<<**extension**>> (p. 354) Mask indicating the types of contents to remove from RTPS frames before saving them to the capture file.*

- class **rti::util::network_capture::TrafficKindMask**

  *<<**extension**>> (p. 354) Mask indicating the traffic direction to capture.*

- class **rti::util::network_capture::NetworkCaptureParams**

  *<<**extension**>> (p. 354) Input parameters for starting Network Capture.*

Functions

- bool **rti::util::network_capture::enable** ()

  Enable Network Capture.

- bool **rti::util::network_capture::disable** ()

  Disable Network Capture.

- bool **rti::util::network_capture::set_default_params** (const **NetworkCaptureParams** &params)

  Set the default Network Capture parameters.

- bool **rti::util::network_capture::start** (const std::string &filename)

  Start Network Capture.

- bool **rti::util::network_capture::start** (**dds::domain::DomainParticipant** participant, const std::string &filename)

  Start Network Capture for a participant.

- bool **rti::util::network_capture::start** (const std::string &filename, const **NetworkCaptureParams** &params)

  Start Network Capture with parameters.

- bool **rti::util::network_capture::start** (**dds::domain::DomainParticipant** participant, const std::string &filename, const **NetworkCaptureParams** &params)

  Start Network Capture with parameters for a participant.

- bool **rti::util::network_capture::stop** ()

  Stop Network Capture.

- bool **rti::util::network_capture::stop** (**dds::domain::DomainParticipant** participant)

  Stop Network Capture.

- bool **rti::util::network_capture::pause** ()

  Pause Network Capture.

- bool **rti::util::network_capture::pause** (**dds::domain::DomainParticipant** participant)

  Pause Network Capture.

- bool **rti::util::network_capture::resume** ()

  Resume Network Capture.

- bool **rti::util::network_capture::resume** (**dds::domain::DomainParticipant** participant)

  Resume Network Capture.

6.205.1 Detailed Description

Save network traffic into a capture file for further analysis.

RTI Connext allows you to capture the network traffic that one or more DomainParticipants send or receive. This feature can be used to analyze and debug communication problems between your DDS applications. When network capture is enabled, each DomainParticipant will generate a pcap-based file that can then be opened by a packet analyzer like Wireshark, provided the right dissectors are installed.

To some extent, network capture can be used as an alternative to existing pcap-based network capture software (such as Wireshark). This will be the case when you are only interested in analyzing the traffic a DomainParticipant sends/receives. In this scenario, network capture will actually have some advantages over using more general pcap-based network capture applications: RTI's network capture includes additional information such as security-related data; it also removes information that is not needed, such as user data, when you want to reduce the capture size. That said, RTI's network capture is not a replacement for other pcap-based network capture applications: it only captures the traffic exchanged by the DomainParticipants, but it does not capture any other traffic exchanged through the system network interfaces.

To capture network traffic **network_capture::enable()** (p. 581) must be invoked before creating any DomainParticipant. Similarly, **network_capture::disable()** (p. 581) must be called after deleting all participants. In between these calls, you may start, stop, pause or resume capturing traffic for one or all participants.

6.205.2 Capturing

Shared Memory Traffic

Every RTPS frame in network capture has a source and a destination associated with it. In the case of shared memory traffic, a process identifier and a port determine the source and destination endpoints.

Access to the process identifier (PID) of the source for inbound traffic requires changes in the shared memory segments. These changes would break shared memory compatibility with previous versions of RTI Connext. For this reason, by default, network capture will not populate the value of the source PID for inbound shared memory traffic.

If interoperability with previous versions of RTI Connext is not necessary, you can generate capture files containing the source PID for inbound traffic. To do so, configure the value of the *'dds.transport.minimum_compatibility_version'* property to 6.1.0. (See **rti::core::policy::Property** (p. 1969)).

```
<domain_participant_qos>

    <property>

        <value>

            <element>

                <name>dds.transport.minimum_compatibility_version</name>

                <value>6.1.0</value>

                <propagate>false</propagate>

            </element>

        </value>

    </property>

</domain_participant_qos>
```

This property is never propagated, so it must be consistently configured throughout the whole system.

Note: Changing the value of this property affects the type of shared memory segments that RTI Connext uses. For that reason, you may see the following warning, resulting from leftover shared memory segments:

```
[0xC733A001,0xB248F671,0xAEC4A0C1:0x000001C1{Domain=200}|CREATE DP|ENABLE]
       NDDS_Transport_Shmem_is_segment_compatible:incompatible shared memory protocol detected.

Current version 4.0 not compatible with 2.0.
```

The leftover shared memory segments can be removed using the `ipcrm` command. See `https://community.rti.com/kb/what-are-possible-solutions-common-shared-memory-issues` for more information.

6.205.3 Function Documentation

6.205.3.1 enable()

```
bool rti::util::network_capture::enable ( )
```

Enable Network Capture.

This method must be called before any other Network Capture method. It must also be called before creating the participants for which we want to capture traffic.

Use this method only for debugging purposes, since it may introduce a significant performance impact.

Returns

true if success. Otherwise, false

MT Safety:

UNSAFE. It is not safe to call this method while another thread may be simultaneously calling another Network Caputre related method, including this one.

See also

network_capture::disable() (p. 581)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

Referenced by **rti::queuing::QueueConsumerParams::enable_availability()**, **rti::queuing::QueueRequester↩ Params::enable_availability()**, **rti::queuing::QueueReplierParams::enable_availability()**, **rti::queuing::↩ QueueRequesterParams::enable_sample_replication()**, **rti::queuing::QueueReplierParams::enable_sample↩ _replication()**, **rti::queuing::QueueProducerParams::enable_wait_for_ack()**, **rti::queuing::QueueRequester↩ Params::enable_wait_for_ack()**, and **rti::queuing::QueueReplierParams::enable_wait_for_ack()**.

6.205.3.2 disable()

```
bool rti::util::network_capture::disable ( )
```

Disable Network Capture.

This method must be the last Network Capture method to be called. It must also be called after deleting the participants for which we captured traffic. Disabling Network Capture without stopping it first is not ok!

Returns

true if success. Otherwise, false

MT Safety:

UNSAFE. It is not safe to call this method while another thread may be simultaneously calling another Network Capture related method, including this one.

See also

network_capture::enable() (p. 581)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

6.205.3.3 set_default_params()

```
bool rti::util::network_capture::set_default_params (
            const  NetworkCaptureParams & params )
```

Set the default Network Capture parameters.

The default parameters are used when Network Capture is started without parameters, i.e., **network_capture::start()** (p. 583).

Precondition

This method requires first enabling Network Capture. See **network_capture::enable()** (p. 581).

Parameters

params	<<***in***>> *(p. 355)*. Configuration parameters that we want to set as defaults.

Returns

true if success. Otherwise, false

See also

network_capture::start() (p. 583)

network_capture::start(dds::domain::DomainParticipant, const std::string &) (p. 583)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

6.205.3.4 start() [1/4]

```
bool rti::util::network_capture::start (
            const std::string & filename )
```

Start Network Capture.

Precondition

This method requires first enabling Network Capture. See **network_capture::enable()** (p. 581).

Parameters

filename	<<***in***>> *(p. 355)*. The name of the output capture file will be based on this input parameter.

In particular, the name for the capture file is the concatenation of the `filename` input parameter, the `"_GUID-"` string followed by the decimal representation of bytes 8-11 of the DomainParticipant's GUID, and the file extension (`".pcap"`).

Returns

true if success. Otherwise, false

See also

network_capture::stop() (p. 586)

network_capture::start(dds::domain::DomainParticipant, const std::string &) (p. 583)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

6.205.3.5 start() [2/4]

```
bool rti::util::network_capture::start (
              dds::domain::DomainParticipant participant,
             const std::string & filename )
```

Start Network Capture for a participant.

Precondition

This method requires first enabling Network Capture. See **network_capture::enable()** (p. 581).

Parameters

participant	<<***in***>> (p. *355*). DomainParticipant for which we want to capture traffic.
filename	<<***in***>> (p. *355*). The name of the output capture file will be based on this input parameter.

In particular, the name for the capture file is the concatenation of the `filename` input parameter, and the file extension (`".pcap"`).

Returns

true if success. Otherwise, false

See also

network_capture::stop() (p. 586)

network_capture::start(dds::domain::DomainParticipant, const std::string &, const NetworkCapture↩ Params &) (p. 585)

network_capture::enable() (p. 581)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

6.205.3.6 start() [3/4]

```
bool rti::util::network_capture::start (
             const std::string & filename,
             const  NetworkCaptureParams & params )
```

Start Network Capture with parameters.

Precondition

This method requires first enabling Network Capture. See **network_capture::enable()** (p. 581).

Perfoms the same function as **network_capture::start()** (p. 583) except that it uses the provided parameters, instead of the default ones.

Parameters

filename	<<***in***>> *(*p. *355)*. The name of the output capture file will be based on this input parameter.

In particular, the name for the capture file is the concatenation of the `filename` input parameter, the `"_GUID-"` string followed by the decimal representation of bytes 8-11 of the DomainParticipant's GUID, and the file extension (`".pcap"`).

Parameters

params	<<***in***>> *(*p. *355)*. Configuration parameters for the capture.

Returns

true if success. Otherwise, false

See also

network_capture::stop() (p. 586)

network_capture::start(dds::domain::DomainParticipant, const std::string &, const NetworkCapture↩ Params &) (p. 585)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

6.205.3.7 start() [4/4]

```
bool rti::util::network_capture::start (
              dds::domain::DomainParticipant participant,
             const std::string & filename,
             const  NetworkCaptureParams & params )
```

Start Network Capture with parameters for a participant.

Precondition

This method requires enabling first Network Capture. See **network_capture::enable()** (p. 581).

Parameters

participant	<<***in***>> *(*p. *355)*. DomainParticipant for which we want to capture traffic.
filename	<<***in***>> *(*p. *355)*. The name of the output capture file will be based on this input parameter.

In particular, the name for the capture file is the concatenation of the `filename` input parameter, and the file extension (`".pcap"`).

Parameters

params	<<***in***>> *(p. 355)*. Parameters for configuring the capture.

Returns

true if success. Otherwise, false

See also

network_capture::stop() (p. 586)

network_capture::start(dds::domain::DomainParticipant, const std::string &) (p. 583)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

6.205.3.8 stop() [1/2]

```
bool rti::util::network_capture::stop ( )
```

Stop Network Capture.

Precondition

This method requires enabling first Network Capture. See **network_capture::enable()** (p. 581).

This method can (and must) be called after **network_capture::start()** (p. 583), not **network_capture::start(dds↩ ::domain::DomainParticipant, const std::string &)** (p. 583). That is, if we start capturing traffic globally (for all DomainParticipants), we must stop capturing traffic also globally. It is not possible to start capturing traffic for a participant but stop it globally.

It is possible to start capturing globally and then stop capturing for a participant, as long as we eventually stop capturing traffic globally.

We must stop capturing for a participant before deleting it.

Returns

true if success. Otherwise, false

See also

network_capture::start() (p. 583)

network_capture::stop() (p. 586)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

6.205.3.9 stop() [2/2]

```
bool rti::util::network_capture::stop (
            dds::domain::DomainParticipant participant )
```

Stop Network Capture.

Precondition

This method requires first enabling Network Capture. See **network_capture::enable()** (p. 581).

Parameters

participant	<<***in***>> *(p. 355).* DomainParticipant for which we want to stop capturing traffic.

Returns

true if success. Otherwise, false

See also

network_capture::start(dds::domain::DomainParticipant, const std::string &) (p. 583)
network_capture::stop() (p. 586)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

6.205.3.10 pause() [1/2]

```
bool rti::util::network_capture::pause ( )
```

Pause Network Capture.

Precondition

This method requires first enabling Network Capture. See **network_capture::enable()** (p. 581).

Returns

true if success. Otherwise, false

See also

network_capture::resume() (p. 588)
network_capture::pause() (p. 587)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

6.205.3.11 pause() [2/2]

```
bool rti::util::network_capture::pause (
            dds::domain::DomainParticipant participant )
```

Pause Network Capture.

Precondition

This method requires first enabling Network Capture. See **network_capture::enable()** (p. 581).

Parameters

participant	<<***in***>> *(p. 355).* DomainParticipant for which we want to pause capturing traffic.

Returns

true if success. Otherwise, false

See also

network_capture::resume() (p. 588)
network_capture::pause() (p. 587)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

6.205.3.12 resume() [1/2]

```
bool rti::util::network_capture::resume ( )
```

Resume Network Capture.

Precondition

This method requires first enabling Network Capture. See **network_capture::enable()** (p. 581).

Returns

true if success. Otherwise, false

See also

network_capture::pause() (p. 587)
network_capture::resume() (p. 588)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

6.205.3.13 resume() [2/2]

```
bool rti::util::network_capture::resume (
              dds::domain::DomainParticipant participant )
```

Resume Network Capture.

Precondition

This method requires first enabling Network Capture. See **network_capture::enable()** (p. 581).

Parameters

participant	<<***in***>> *(p. 355).* DomainParticipant for which we want to resume capturing traffic.

Returns

true if success. Otherwise, false

See also

network_capture::pause() (p. 587)

network_capture::resume() (p. 588)

Note

This is a standalone function in the namespace **rti::util::network_capture** (p. 780)

6.206 Heap Monitoring

Monitor memory allocations done by the middleware on the native heap.

Classes

- struct **rti::util::heap_monitoring::SnapshotOutputFormat_def**

  Specify the format of the output of the snapshot. RTI Connext.
- struct **rti::util::heap_monitoring::SnapshotContentFormat_def**

  Bitmap used to decide which information of the snapshot will be displayed.
- class **rti::util::heap_monitoring::HeapMonitoringParams**

  Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.

Typedefs

- typedef **dds::core::safe_enum**< **SnapshotOutputFormat_def** > **rti::util::heap_monitoring::Snapshot↩ OutputFormat**

  Specify the format of the output of the snapshot.
- typedef **dds::core::safe_enum**< **SnapshotContentFormat_def** > **rti::util::heap_monitoring::Snapshot↩ ContentFormat**

  Bitmap used to decide which information of the snapshot will be displayed.

Functions

- bool **rti::util::heap_monitoring::enable** ()

  Starts monitoring the heap memory used by RTI Connext.
- bool **rti::util::heap_monitoring::enable** (const **HeapMonitoringParams** &params)

  Starts monitoring the heap memory used by RTI Connext with params.
- void **rti::util::heap_monitoring::disable** ()

  Stops monitoring the heap memory used by RTI Connext.
- bool **rti::util::heap_monitoring::pause** ()

  Pauses heap monitoring.
- bool **rti::util::heap_monitoring::resume** ()

  Resumes heap monitoring.
- bool **rti::util::heap_monitoring::take_snapshot** (const std::string &filename, bool print_details=false)

  Saves the current heap memory usage in a file.

6.206.1 Detailed Description

Monitor memory allocations done by the middleware on the native heap.

RTI Connext allows you to monitor the memory allocations done by the middleware on the native heap. This feature can be used to analyze and debug unexpected memory growth.

After heap_monitoring::enable() is called, you may invoke **heap_monitoring::take_snapshot()** (p. 593) to save the current heap memory usage to a file. By comparing two snapshots, you can tell if new memory has been allocated and, in many cases, where.

6.206.2 Typedef Documentation

6.206.2.1 SnapshotOutputFormat

typedef **dds::core::safe_enum**< **SnapshotOutputFormat_def**> **rti::util::heap_monitoring::Snapshot↩ OutputFormat**

Specify the format of the output of the snapshot.

See also

SnapshotOutputFormat_def (p. 2371)

6.206.2.2 SnapshotContentFormat

typedef **dds::core::safe_enum**< **SnapshotContentFormat_def**> **rti::util::heap_monitoring::Snapshot↩ ContentFormat**

Bitmap used to decide which information of the snapshot will be displayed.

See also

SnapshotContentFormat_def (p. 2370)

6.206.3 Function Documentation

6.206.3.1 enable() [1/2]

bool rti::util::heap_monitoring::enable ( )

Starts monitoring the heap memory used by RTI Connext.

This function must be called before any other function in the RTI Connext library is called.

Once heap monitoring is enabled, you can take heap snapshots by using **heap_monitoring::take_snapshot()** (p. 593).

Use this method only for debugging purposes, since it may introduce a significant performance impact.

MT Safety:

UNSAFE. It is not safe to call this method while another thread may be simultaneously calling another heap-related method, including this one.

Returns

true if success. Otherwise, false

See also

heap_monitoring::disable() (p. 592)

Note

This is a standalone function in the namespace **rti::util::heap_monitoring** (p. 779)

6.206.3.2 enable() [2/2]

```
bool rti::util::heap_monitoring::enable (
            const  HeapMonitoringParams & params )
```

Starts monitoring the heap memory used by RTI Connext with params.

Perfoms the same function as **heap_monitoring::enable()** (p. 591) except that it also provides the values in params. Those values will set the format used in the snapshot **heap_monitoring::take_snapshot()** (p. 593).

Returns

true if success. Otherwise, false

See also

heap_monitoring::disable() (p. 592)

Note

This is a standalone function in the namespace **rti::util::heap_monitoring** (p. 779)

6.206.3.3 disable()

```
void rti::util::heap_monitoring::disable ( )
```

Stops monitoring the heap memory used by RTI Connext.

This method must be the last method called from RTI Connext.

See also

heap_monitoring::enable() (p. 591)

Note

This is a standalone function in the namespace **rti::util::heap_monitoring** (p. 779)

6.206.3.4 pause()

```
bool rti::util::heap_monitoring::pause ( )
```

Pauses heap monitoring.

New memory allocations will not be monitored and they will not appear in the snapshot generated by **heap_monitoring::take_snapshot()** (p. 593).

Returns

true if success. Otherwise, false

See also

heap_monitoring::resume() (p. 593)

Note

This is a standalone function in the namespace **rti::util::heap_monitoring** (p. 779)

6.206.3.5 resume()

```
bool rti::util::heap_monitoring::resume ( )
```

Resumes heap monitoring.

Returns

true if success. Otherwise, false

See also

heap_monitoring::pause() (p. 592)

Note

This is a standalone function in the namespace **rti::util::heap_monitoring** (p. 779)

6.206.3.6 take_snapshot()

```
bool rti::util::heap_monitoring::take_snapshot (
             const std::string & filename,
             bool print_details = false )
```

Saves the current heap memory usage in a file.

After **heap_monitoring::enable()** (p. 591) is called, you may invoke this method periodically to save the current heap memory usage to a file.

By comparing two snapshots, you can tell if new memory has been allocated and in many cases where. This is why this operation can be used to debug unexpected memory growth.

The format of a snapshot is as follows:

First, there is a memory usage summary like this:

```
<P>
                    Product Version: NDDSCORE_BUILD_6.0.0.0_20200316T123411Z_RTI_ENG

                    Process virtual memory: 2552352768

                    Process physical memory: 16187392

                    Current application heap usage: 10532131

                    Approximate total heap usage: 203331110

                    High watermark: 10532131

                    Alloc count: 17634

                    Free count: 3518
```

- Process virtual memory: The amount of virtual memory in bytes taken by the process. This memory includes RTI Connext and non-RTI Connext memory.
- Process virtual memory: The amount of physical memory in bytes taken by the process.
- Current application heap usage: The amount of heap memory in bytes used by the middleware. For Java and .NET APIs, this memory only accounts for unmanaged RTI Connext memory, not memory living in the managed heap. This value does not include overhead memory allocations that are used by the Heap Monitoring utility. It therefore provides the heap usage that is used when Heap Monitoring is disabled and does not reflect the actual amount of memory that has been allocated by the middleware. That value is accounted for in 'Approximate total heap usage'.
- Approximate total heap usage: The amount of heap memory in bytes used by the middleware, including overhead allocations from the Heap Monitoring utility. When the Heap Monitoring utility is enabled, every allocation has an additional overhead number of bytes allocated so that the middleware can keep track of the meta-data that is output in the heap snapshots. This overhead is not accounted for in the 'Current application heap usage' summary field, but is included in this field. For Java and .NET APIs, this memory only accounts for unmanaged RTI Connext memory, not memory living in the managed heap.
- High watermark: The maximum amount of heap usage by RTI Connext since **heap_monitoring::enable()** (p. 591) was invoked.
- Alloc count: The number of invocations to malloc, realloc, or calloc operations done by RTI Connext.
- Free count: The number of invocations to the free operation done by RTI Connext.

After the previous summary, and only if you set the parameter print_details to true, the method will print the details of every single outstanding heap allocation done by RTI Connext. For example:

```
<P>
                block_id, timestamp, block_size, alloc_method_name, type_name, pool_alloc, pool_buffer_size,
    pool_buffer_count, topic_name, function_name, activity_context

                23087, 1586943520, 16, RTIOsapiHeap_allocateArray, struct RTIEncapsulationInfo, MALLOC, 0,
    0, PRESServiceRequest, PRESWriterHistoryDriver_new,
    "0X101175A,0X76DD63D7,0X984377BC:0X1C1{Name=ShapeTypeParticipant,Domain=110}|CREATE
    Participant|ENABLE|:0X80000088{Entity=Pu,Domain=110}|CREATE Writer WITH TOPIC PRESServiceRequest"
```

- block_id: Block ID of the allocation. This number increases with every allocation.
- timestamp: Timestamp in UTC seconds corresponding to the time where the allocation was done.
- block_size: The number of bytes allocated.
- alloc_method_name: The allocation RTI Connext method name.
- type_name: The allocation typename.
- pool_alloc: Indicates if the heap allocation is a RTI Connext pool allocation (POOL) or a regular allocation (MALLOC).
- pool_buffer_size: For pool allocations, this number indicates the size of the elements in the pool in number of bytes. block_size is equal to (pool_buffer_size ∗ pool_buffer_count).
- pool_buffer_count: For pool allocations, this number indicates the number of buffers allocated for the pool. block↩ _size is equal to (pool_buffer_size ∗ pool_buffer_count).
- topic_name: The topic name associated with the allocation or 'n/a' if it is not available.
- function_name: function name associated with the allocation or 'n/a' if it is not available.
- activity_context: **Activity Context** (p. 449)

Parameters

filename	<<***in***>> *(p. 355)*. Name of file in which to store the snapshot.
print_details	<<***in***>> (p. *355)*. Indicates if the snapshot will contain only the memory usage summary or the details of the individual allocations.

Returns

true if success. Otherwise, false

Note

This is a standalone function in the namespace **rti::util::heap_monitoring** (p. 779)

6.207 Other Utilities

Other Utilities, such as **rti::util::spin()** (p. 597)

Functions

- void **rti::config::enable_automatic_tss_cleanup** (bool enable)

  Enable automatic thread-specific storage (TSS) cleanup on platforms that support it.
- void **rti::util::sleep** (const **dds::core::Duration** &durationIn)

  Blocks the calling thread for the specified duration.
- uint64_t **rti::util::spin_per_microsecond** ()

  Returns the number of spin operations needed to wait 1 microsecond.
- void **rti::util::spin** (uint64_t spin_count)

  Performs a spin operation (active wait) as many times as indicated.
- bool **rti::util::activation::unload_activation_library** ()

  Unloads the Activation Library (rti_activation) if it was previously loaded.

6.207.1 Detailed Description

Other Utilities, such as **rti::util::spin()** (p. 597)

6.207.2 Function Documentation

6.207.2.1 enable_automatic_tss_cleanup()

```
void rti::config::enable_automatic_tss_cleanup (
            bool enable )
```

Enable automatic thread-specific storage (TSS) cleanup on platforms that support it.

This API must be called before any other RTI Connext API.

Setting it to one value and then setting it to the opposite value after any RTI Connext API has been called may result in undefined behavior.

See also

DomainParticipantFactory::unregister_thread

Note

This is a standalone function in the namespace rti::util::config

6.207.2.2 sleep()

```
void rti::util::sleep (
            const  dds::core::Duration & durationIn )
```

Blocks the calling thread for the specified duration.

Note that the achievable resolution of sleep is OS-dependent. That is, do not assume that you can sleep for 1 nanosecond just because you can specify a 1-nanosecond sleep duration via the API. The sleep resolution on most operating systems is usually 10 ms or greater.

Parameters

duration↩ In	<<***in***>> *(*p. *355)* Sleep duration.

MT Safety:

safe

Examples

Foo_publisher.cxx.

6.207.2.3 spin_per_microsecond()

```
uint64_t rti::util::spin_per_microsecond ( )
```

Returns the number of spin operations needed to wait 1 microsecond.

This utility can be used to measure how many spin operations must be performed to wait 1 microsecond. Since the time that it takes the CPU to perform 1 spin operation depends on the CPU frequency, it is recommended to use this utility before using **spin()** (p. 597).

Returns

Number of spin operations to wait 1 microsecond.

See also

rti::util::spin() (p. 597)

6.207.2.4 spin()

```
void rti::util::spin (
             uint64_t spin_count )
```

Performs a spin operation (active wait) as many times as indicated.

Spinning is the action of performing useless operations in a for loop in order to actively wait some time without yielding the CPU. Given that the resolution of sleep is in the order of ms, you can use this utility to wait times in the order of microseconds. To properly use this functionality, it is useful to measure previously the number of spin operations needed to wait the equivalent to microsecond (using the utility get_spin_per_microsecond) and then compute the corresponding spin count desired.

Parameters

spin_count	<<***in***>> *(p. 355)* Number of spin operations to perform.

See also

rti::util::spin_per_microsecond() (p. 597)

6.207.2.5 unload_activation_library()

```
bool rti::util::activation::unload_activation_library ( )
```

Unloads the Activation Library (rti_activation) if it was previously loaded.

This function should be called just before your process finishes. See the License Management section in the `Installation Guide` for additional details.

Returns

true if successful. Otherwise, false

6.208 Monitoring Library 2.0

RTI Monitoring Library 2.0 public API.

Functions

- void **RTI_Monitoring_initialize** (void)
  Initializes Monitoring Library 2.0.

Variables

- const char ∗const **RTI_MONITORING_PERIODIC_TOPIC_NAME**
  The name of the Monitoring Topic used for periodic metrics distribution.
- const char ∗const **RTI_MONITORING_EVENT_TOPIC_NAME**
  The name of the Monitoring Topic used for event metrics distribution.
- const char ∗const **RTI_MONITORING_LOGGING_TOPIC_NAME**
  The name of the Monitoring Topic used for log messages distribution.

6.208.1 Detailed Description

RTI Monitoring Library 2.0 public API.

RTI Monitoring Library 2.0 is one component of the RTI Connext Observability Framework which allows collecting and distributing telemetry data (metrics and logs) associated with the observable resources created by an RTI Connext application.

In this release, the only Observable resources are the following entities: **dds::pub::DataWriter** (p. 1138), **dds::sub↩ ::DataReader** (p. 976), **dds::pub::Publisher** (p. 1993), **dds::sub::Subscriber** (p. 2410), **dds::domain::Domain↩ Participant** (p. 1326), **dds::topic::Topic** (p. 2474) and Application (a process running RTI Connext).

The library also accepts remote commands to change the set of distributed telemetry data at run-time.

The data distributed by RTI Monitoring Library 2.0 is sent to an RTI Observability Collector Service instance, which forwards the data to other RTI Observability Collector Service instances or stores the data in third-party observability backends such as Prometheus or Grafana Loki.

RTI Monitoring Library 2.0 is a separate library (rtimonitoring2), and applications can use it in three different modes:

- **Dynamically loaded:** This is the default mode, and it requires that the rtimonitoring2 shared library is in the library search path.
- **Dynamic linking:** The application is linked with the rtimonitoring2 shared library.
- **Static linking:** The application is linked with the rtimonitoring2 static library.

The last two modes require calling the API **RTI_Monitoring_initialize** (p. 599) in your application before any other RTI Connext APIs. This API is defined in the header file `ndds/monitoring/monitoring_monitoringClass.h`.

Dynamic and static linking are only supported in C and C++ applications.

To enable use of RTI Monitoring Library 2.0 and configure its behavior, use the **rti::core::policy::Monitoring** (p. 1706) QoS policy on the **dds::domain::DomainParticipant::participant_factory_qos()** (p. 1340). This QoS policy can be configured programmatically or via XML.

6.208.2 Function Documentation

6.208.2.1 RTI_Monitoring_initialize()

```
void RTI_Monitoring_initialize (
             void  )
```

Initializes Monitoring Library 2.0.

This method must be called before calling any other RTI Connext API if the application links with RTI Monitoring Library 2.0 dynamically or statically.

6.208.3 Variable Documentation

6.208.3.1 RTI_MONITORING_PERIODIC_TOPIC_NAME

```
const char* const RTI_MONITORING_PERIODIC_TOPIC_NAME
```

The name of the Monitoring Topic used for periodic metrics distribution.

String version: "DCPSPeriodicStatusMonitoring".

This topic is used internally by Monitoring Library 2.0 and RTI Observability Collector Service. You should not publish or subscribe to it.

6.208.3.2 RTI_MONITORING_EVENT_TOPIC_NAME

```
const char* const RTI_MONITORING_EVENT_TOPIC_NAME
```

The name of the Monitoring Topic used for event metrics distribution.

String version: "DCPSEventStatusMonitoring".

This topic is used internally by Monitoring Library 2.0 and RTI Observability Collector Service. You should not publish or subscribe to it.

6.208.3.3 RTI_MONITORING_LOGGING_TOPIC_NAME

```
const char* const RTI_MONITORING_LOGGING_TOPIC_NAME
```

The name of the Monitoring Topic used for log messages distribution.

String version: "DCPSLoggingStatusMonitoring".

This topic is used internally by Monitoring Library 2.0 and RTI Observability Collector Service. You should not publish or subscribe to it.

6.209 Built-in Types Examples

Using Built-in Types.

Using Built-in Types.

RTI Connext provides a set of simple data types for you to use with the topics in your application (see **Built-in Types** (p. 50))

If you use these built-in types you do not have to generate any code using rtiddsgen.

- StringTopicType has one field, `data`, of type **dds::core::string** (p. 435).

```
dds::domain::DomainParticipant participant(0);
dds::topic::Topic<dds::core::StringTopicType> topic(participant, "MyTopic");
dds::pub::DataWriter<dds::core::StringTopicType> writer(
   dds::pub::Publisher(participant), topic);
// Create a StringTopicType sample with data = "Example Data"
dds::core::StringTopicType sample("Example Data");
// Change the value of data
sample.data("New Data Value");
// Write the sample
writer.write(sample);
// You can also write a sample like this, making use of an implicit
// conversion between string and StringTopicType
writer.write("Example Data 2");
```

- KeyedStringTopicType has two fields, `key` and `value`, both of type **dds::core::string** (p. 435).

```
dds::domain::DomainParticipant participant(0);
dds::topic::Topic<dds::core::KeyedStringTopicType> topic(
   participant, "MyTopic");
dds::pub::DataWriter<dds::core::KeyedStringTopicType> writer(
   dds::pub::Publisher(participant), topic);
// Create a StringTopicType sample with data = "Example Data"
dds::core::KeyedStringTopicType sample("Example Key", "Example Value");
// Change the value of value and key
sample.key("New Key");
sample.value("New Value");
// Write the sample
writer.write(sample);
```

- BytesTopicType represents an array of bytes:

```
dds::domain::DomainParticipant participant(0);
dds::topic::Topic<dds::core::BytesTopicType> topic(participant, "MyTopic");
dds::pub::DataWriter<dds::core::BytesTopicType> writer(
    dds::pub::Publisher(participant), topic);
// Create a BytesTopicType sample with some data
std::vector<uint8_t> byte_vector;
byte_vector.push_back(100);
byte_vector.push_back(200);
dds::core::BytesTopicType sample(byte_vector);
// You can modify it with the operator[]:
sample[0] = 150;
// Write the sample
writer.write(sample);
// You can also write the std::vector directly, using the implicit
// BytesTopicType constructor:
writer.write(byte_vector);
```

6.210 Exceptions

How DDS return codes map to C++ exceptions.

How DDS return codes map to C++ exceptions.

6.210.1 Exceptions

The following is a list of the exceptions that may be thrown by RTI Connext. They are a part of the **dds::core** (p. 616) namespace and can also be found in hpp/dds/core/Exception.hpp. The ∗Error classes can be caught simply by catching an std::exception.

modern C++ API Exception Class	DDS PIM Return Code	Std C++ Parent Exception
Normal return; no exception	RETCODE_OK	N/A
An informational state attached to a normal return; no exception	RETCODE_NO_DATA	N/A
Error	RETCODE_ERROR	std::logic_error
InvalidArgumentError	RETCODE_BAD_PARAMETER	std::invalid_argument
TimeoutError	RETCODE_TIMEOUT	std::runtime_error
UnsupportedError	RETCODE_UNSUPPORTED	std::logic_error
AlreadyClosedError	RETCODE_ALREADY_DELETED	std::logic_error
IllegalOperationError	RETCODE_ILLEGAL_OPERATION	std::logic_error
NotEnabledError	RETCODE_NOT_ENABLED	std::logic_error
PreconditionNotMetError	RETCODE_PRECONDITION_NOT_↩ MET	std::logic_error
ImmutablePolicyError	RETCODE_IMMUTABLE_POLICY	std::logic_error
InconsistentPolicyError	RETCODE_INCONSISTENT_POLICY	std::logic_error
NotAllowedBySecurityError	RETCODE_NOT_ALLOWED_BY_↩ SECURITY	std::logic_error
OutOfResourcesError	RETCODE_OUT_OF_RESOURCES	std::runtime_error
InvalidDowncastError	N/A	std::runtime_error
NullReferenceError	N/A	std::runtime_error
InvalidDataError	N/A	std::logic_error

6.211 Qos Use Cases

The different QoS classes are containers of QoS policies of a concrete **dds::core::Entity** (p. 1520). Each Entity has a qos() getter and setter.

- **dds::domain::qos::DomainParticipantQos** (p. 1384) is the container of the **dds::domain::DomainParticipant** (p. 1326) QoS policies.
- **dds::topic::qos::TopicQos** (p. 2509) is the container of the **dds::topic::Topic** (p. 2474) QoS policies.
- **dds::pub::qos::PublisherQos** (p. 2007) is the container of the **dds::pub::Publisher** (p. 1993) QoS policies.
- **dds::sub::qos::SubscriberQos** (p. 2423) is the container of the **dds::sub::Subscriber** (p. 2410) QoS policies.
- **dds::pub::qos::DataWriterQos** (p. 1226) is the container of the **dds::pub::DataWriter** (p. 1138) QoS policies.
- **dds::sub::qos::DataReaderQos** (p. 1072) is the container of the **dds::sub::DataReader** (p. 976) QoS policies.

6.211.1 Setting Qos Values

There are a few different ways to set the values of a QoS policy within an entity's QoS object

- Setting a QoS policy with operator <<

```
using namespace dds::core::policy;
// Construct a Reliability QoS object with default values
dds::core::policy::Reliability reliability;
// Construct a DataReaderQos object with default values
dds::sub::qos::DataReaderQos reader_qos;
// Set the Reliability Qos policy field of the DataReaderQos using operator «,
// setting the ReliabilityKind to RELIABLE inline
reader_qos « reliability.kind(ReliabilityKind::RELIABLE);
```

- Many of the QoS policies have appropriately named constructors that will construct a QoS policy object with values for some of the fields in the policy. The following code example is equivalent to the above example.

```
using namespace dds::core::policy;
// Use a named constructor to set the ReliabilityKind
reader_qos « Reliability::Reliable();
```

- You can also set a QoS policy on a QoS object using the QoS object's policy() setter which is templatized on the QoS policy that it is being used to set. The following code example is equivalent to the above two examples.

```
using namespace dds::core::policy;
// Set the reliability Qos on the DataReaderQos object
reader_qos.policy<Reliability>(Reliability::Reliable());
// Note that the above statement sets the whole policy, possibly overriding
// other fields of Reliability (such as Reliability::acknowledgment_kind())
// with their default values. If you only want to change a specific field
// of the policy, get it by reference and modify it:
reader_qos.policy<Reliability>().kind(ReliabilityKind::RELIABLE);
```

6.211.2 Getting Qos Values

There are a few different ways to get the values of a QoS policy from an entity's QoS object

- Getting a QoS policy with operator >>

```
using namespace dds::core::policy;
// Construct a Reliability QoS object with default values
dds::core::policy::Reliability reliability;
// Construct a DataReaderQos object with default values
dds::sub::qos::DataReaderQos reader_qos;
// Populate the Reliability Qos object with the values from the
// DataReaderQos object using operator »
reader_qos » reliability;
```

- You can also get a QoS policy from a QoS object using the QoS object's policy() getter which is templatized on the QoS policy that it is being used to get. The following code example is equivalent to the above example.

```
using namespace dds::core::policy;
// Get the reliability Qos from the DataReaderQos object
reliability = reader_qos.policy<Reliability>();
```

6.212 Qos Provider Use Cases

How to use **dds::core::QosProvider** (p. 2025) to access XML QoS profiles.

How to use **dds::core::QosProvider** (p. 2025) to access XML QoS profiles.

6.212.1 Managing Qos Profiles

This section provides several examples on how to use a **dds::core::QosProvider** (p. 2025) to load QoS profiles from XML. The class documentation provides general usage information.

These examples use the following file, `ExampleQos.xml`, which defines only the `EntityName` QoS policy for the `DomainParticipant` and `DataWriter` to illustrate how the profiles are loaded:

```
<?xml version="1.0"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="https://community.rti.com/schema/current/rti_dds_profiles.xsd">
    <qos_library name="MyLibrary">
        <qos_profile name="MyProfile" is_default_qos="true">
            <domain_participant_qos>
                <participant_name>
                    <name>ExampleParticipantName</name>
                </participant_name>
            </domain_participant_qos>
            <datawriter_qos>
                <publication_name>
                    <name>ExamplePublicationName</name>
                </publication_name>
            </datawriter_qos>
        </qos_profile>
    </qos_library>
    <qos_library name="MySecondLibrary">
        <qos_profile name="MySecondProfile">
            <domain_participant_qos>
                <participant_name>
                    <name>SecondExampleParticipantName</name>
                </participant_name>
            </domain_participant_qos>
            <datawriter_qos>
                <publication_name>
                    <name>SecondExamplePublicationName</name>
                </publication_name>
            </datawriter_qos>
        </qos_profile>
    </qos_library>
</dds>
```

The example code shows how to create a QosProvider, load QoS profiles from the provider, and create entities with those QoS values:

```
    // Load the contents of an XML file:
    dds::core::QosProvider my_provider("ExampleQos.xml");
    // Create a participant with the default profile:
    auto participant_qos = my_provider.participant_qos();
    dds::domain::DomainParticipant participant(0, participant_qos);
    // This prints ExampleParticipantName because MyLibrary::MyProfile is marked
    // with is_default_qos="true"
    std::cout « participant.qos().policy<EntityName>().name().value()
              « std::endl;
    // Load a specific profile. This time it prints SecondExampleParticipantName
    participant_qos =
            my_provider.participant_qos("MySecondLibrary::MySecondProfile");
    std::cout « participant_qos.policy<EntityName>().name().value()
              « std::endl;
    // Change the default profile:
    my_provider.extensions().default_profile("MySecondLibrary::MySecondProfile");
    // Now when the profile is not specified, participant_qos() looks at
    // "MySecondLibrary::MySecondProfile", printing SecondExampleParticipantName
    participant_qos = my_provider.participant_qos();
    std::cout « participant_qos.policy<EntityName>().name().value()
              « std::endl;
```

```
// Create a topic, publisher and writer:
dds::topic::Topic<Foo> topic(participant, "Example Foo");
dds::pub::Publisher publisher(participant);
dds::pub::DataWriter<Foo> writer(
        publisher,
        topic,
        my_provider.datawriter_qos());
// Prints SecondExamplePublicationName
std::cout << writer.qos().policy<EntityName>().name().value()
          << std::endl;
// If you don't want to specify the QoS argument for each writer you create
// you can set the default value:
publisher.default_datawriter_qos(
        my_provider.datawriter_qos("MyLibrary::MyProfile"));
dds::pub::DataWriter<Foo> other_writer(publisher, topic);
// Prints ExamplePublicationName
std::cout << writer.qos().policy<EntityName>().name().value() << std::endl;
// You can also load the QoS profiles from a string:
const char * my_xml_str =
    "str://\"<dds>"
        "<qos_library name=\"MyThirdLibrary\">"
            "<qos_profile name=\"MyThirdProfile\">"
                "<domain_participant_qos>"
                    "<participant_name>"
                        "<name>ThirdExampleParticipantName</name>"
                    "</participant_name>"
                "</domain_participant_qos>"
            "</qos_profile>"
        "</qos_library>"
    "</dds>\"";
// In this case we're also specifying which profile we want to load by
// default in the constructor
dds::core::QosProvider my_provider2(
        my_xml_str,
        "MyThirdLibrary::MyThirdProfile");
// This prints ThirdExampleParticipantName
participant_qos = my_provider2.participant_qos();
std::cout << participant_qos.policy<EntityName>().name().value()
          << std::endl;
// You can also examine which libraries and profiles that have been
// loaded:
auto libraries = my_provider.extensions().qos_profile_libraries();
for (const std::string& library : libraries) {
    std::cout << library << std::endl;
    auto profiles = my_provider.extensions().qos_profiles(library);
    for (const std::string& profile : profiles) {
        std::cout << " -" << profile << std::endl;
    }
}
```

See also

Create a DynamicType from an XML description (p. 610)

XML Application Creation (p. 342)

6.212.2 The Default Qos Provider

This example shows how to configure the default QosProvider:

```
// We're going to configure the default QoS Provider to load ExampleQos.xml
// and to ignore the NDDS_QOS_PROFILES environment variable and the file
// USER_QOS_PROFILES.xml
rti::core::QosProviderParams params;
params.url_profile({ "ExampleQos.xml" });
params.ignore_environment_profile(true);
params.ignore_user_profile(true);
// To ensure that the new configuration takes effect before any other
// profiles are loaded, set the new parameters before accessing
// QosProvider::Default().
rti::core::default_qos_provider_params(params);
dds::core::QosProvider default_provider = dds::core::QosProvider::Default();
dds::domain::DomainParticipant participant1(0, default_provider.participant_qos());
// Prints "ExampleParticipantName"
std::cout << participant1.qos().policy<EntityName>().name().value()
```

```
            « std::endl;
    // Change the default profile
    default_provider.extensions().default_profile("MySecondLibrary::MySecondProfile");
    dds::domain::DomainParticipant participant2(1, default_provider.participant_qos());
    // Prints "SecondExampleParticipantName"
    std::cout « participant2.qos().policy<EntityName>().name().value()
            « std::endl;
    // Reset the QosProviderParams for the default QosProvider
    dds::core::QosProvider::reset_default();
    // This participant will not have a participant name (the default
    // RTI Connext value) because we reset the default QosProviderParams,
    // meaning that the value for string_profile is empty and the above example
    // XML Qos profile is not loaded.
    dds::domain::DomainParticipant participant3(2, default_provider.participant_qos());
    // Prints false
    std::cout « std::boolalpha
            « participant2.qos().policy<EntityName>().name().has_value()
            « std::endl;
```

6.213 Working with IDL types

How IDL types map to C++ classes.

Classes

- class **Foo**

  An example topic-type.

6.213.1 Detailed Description

How IDL types map to C++ classes.

6.213.2 Example IDL types

The following IDL code defines the types we will use in these examples.

```
struct Foo {
    long x; //@key
    long y;
};
struct MyType {
    long my_long;
    string<512> my_string;
    Foo my_foo;
    sequence<long, 10> my_sequence;
    Foo my_array[5];
    Foo my_optional; //@Optional
};
```

To generate C++ code for these types, see the `Code Generator User's Manual`.

6.213.3 Constructors and operators

C++ types generated from IDL have **value semantics** (p. 350) and provide a default constructor, a copy constructor, a move constructor <<***C++11***>> *(p. 353)*, a constructor with parameters to set all the type's members, a destructor, a copy-assignment operator, and a move-assignment operator <<***C++11***>> *(p. 353)*. Types also include equality operators, the operator << to `std::ostream` <<***extension***>> *(p. 354)* and a namespace-level `swap` function.

```
// The default constructor recursively initializes all members of a sample
MyType default_sample;
// A copy constructor is also provided
MyType copied_sample = default_sample;
// For convenience, another constructor allows initializing all the members
MyType initialized_sample (
    7, // my_long
    std::string("Hello, World!"), // my_string
    Foo(1, 2), // my_foo (initializing its members 'x' and 'y')
    std::vector<int32_t>(3, 2), // my_sequence containing 3 times the value 2
    dds::core::array<Foo, 5>(), // my_array
    Foo(2, 3) // my_optional (initialized with a non-empty value)
);
// An assignment operator is also available
copied_sample = initialized_sample;
// Equals operator (!= also available)
if (copied_sample == initialized_sample) {
    std::cout « "Samples are equal\n";
}
// You can move a value into a member:
std::vector<int32_t> tmp_vector {1, 2, 3, 4};
copied_sample.my_sequence() = std::move(tmp_vector);
std::cout « "You can print samples to an output stream: "
          « copied_sample
          « std::endl;
```

In addition to that, a number of **traits** (p. 444) provide additional information and utilities for IDL-generated types.

6.213.4 Accessing the type members

Struct members are public. Setters and getters provide access to union members.

The following table summarizes how different IDL types map to C++.

IDL type	C++ type
`string`	`std::string`
bounded sequence (`sequence<T, M>`)	**rti::core::bounded_sequence**<**T, M**> (p. 887)
unbounded sequence (`sequence<T>`)	`std::vector<T>`
array (`T member`[N])	std::array<T, N>
optional member (`@optional T member;`)	**dds::core::optional**<**T**> (p. 1879)
external member (`@external T member;`)	**dds::core::external**<**T**> (p. 1551)

The type **`rti::core::bounded_sequence`** (p. 887) offers similar functionality to that of a `std::vector`. But because it has an upper bound `M`, it provides two advantages:

1) A better memory management strategy for deserializing data samples in a DataReader, improving the overall performance of the middleware.

2) Its member functions check for out-of-bounds growth.

It is possible to use `std::vector` for bounded sequences by annotating them with `@use_vector` in IDL. For example:

```
struct SequenceExample {
    @use_vector sequence<long, 10> bounded_sequence_as_vector;
    sequence<long, 10> bounded_sequence;
    sequence<long> unbounded_sequence;
};
```

The type `dds::core::array` is just an alias of `std::array` if available, or an alias of `boost::array` otherwise.

6.213.5 Enumerations

IDL enums map to `enum class` types in C++.

6.213.6 Unions

IDL unions map to C++ classes with getters and setters for each member, plus a especial getter and setter for the discriminator, `_d()`. At any moment only one member, the one selected by the discriminator, is considered active. A member getter will throw **dds::core::PreconditionNotMetError** (p. 1941) if the discriminator selects a different member.

Member setters set the correct discriminator value automatically.

The default constructor default-constructs all the members and selects the discriminator returned by the static member function `default_discriminator()`.

The following example shows an IDL union and C++ code to manipulate it.

- IDL definition:

```
enum MyUnionDiscriminator {
    USE_LONG,
    USE_STRING,
    USE_FOO
};
union MyUnion switch (MyUnionDiscriminator) {
case USE_LONG:
    long my_long;
case USE_STRING:
    string my_string;
case USE_FOO:
    Foo my_foo;
};
```

- C++ example:

```
    MyUnion my_sample;
    // Set a member
    my_sample.my_string("Hello, World!");
    // The discriminator is automatically updated
    assert (my_sample._d() == MyUnionDiscriminator::USE_STRING);
    // Access that member
    std::cout « my_sample.my_string() « std::endl;
    // Trying to access my_long() will throw an exception:
    try {
        std::cout « my_sample.my_long() « std::endl;
    } catch (const dds::core::PreconditionNotMetError&) {
        std::cout « "my_long is not selected!\n";
    }
    // First, set it
    my_sample.my_long(4);
    std::cout « my_sample.my_long() « std::endl;
    // Trying to access my_foo() to modify a nested member will also fail
    try {
        my_sample.my_foo().x(3);
    } catch (const dds::core::PreconditionNotMetError&) {
        std::cout « "my_foo is not selected!\n";
    }
    // First change the discriminator
    my_sample._d(MyUnionDiscriminator::USE_FOO);
    my_sample.my_foo().x(3); // now it works
    std::cout « my_sample « std::endl;
```

6.214 DynamicType and DynamicData Use Cases

Using DynamicType and DynamicData.

Using DynamicType and DynamicData.

The following #includes are needed for the examples on this page

```
#include <dds/core/ddscore.hpp>
#include <dds/pub/ddspub.hpp>
#include <dds/sub/ddssub.hpp>
#include "Foo.hpp"
```

6.214.1 Creating a DynamicType

In this section we'll see the different ways to create a **dds::core::xtypes::DynamicType** (p. 1504) that represents the following **IDL type** (p. 606) `MyType` **:**

```
struct Foo {
    long x; //@key
    long y;
};
struct MyType {
    long my_long;
    string<512> my_string;
    Foo my_foo;
    sequence<long, 10> my_sequence;
    Foo my_array[5];
    Foo my_optional; //@Optional
};
```

6.214.1.1 Create a DynamicType by instantiating it and adding members

(Note this example uses C++11 code that can easily be converted to similar C++03 code)

```
dds::core::xtypes::StructType create_mytype()
{
    using namespace dds::core::xtypes;
    // First, let's create Foo. We're using an initializer_list of Members
    StructType foo(
        "Foo", {
            Member("x", primitive_type<int32_t>()).key(true),
            Member("y", primitive_type<int32_t>())
        }
    );
    // Create MyType and add members
    StructType mytype("MyType");
    mytype.add_member(Member("my_long", primitive_type<int32_t>()));
    mytype.add_member(Member("my_string", StringType(512)));
    mytype.add_member(Member("my_foo", foo));
    mytype.add_member(Member("my_sequence", SequenceType(primitive_type<int32_t>(), 10)));
    mytype.add_member(Member("my_array", ArrayType(foo, 5)));
    mytype.add_member(Member("my_optional", foo).optional(true));
    rti::core::xtypes::print_idl(mytype);
    return mytype;
}
```

6.214.1.2 Create a DynamicType from an XML description

We can define the type in this XML file, MyType.xml:

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="https://community.rti.com/schema/current/rti_dds_profiles.xsd">
    <types>
        <struct name= "Foo">
            <member name="x" type="int32" key="true"/>
            <member name="y" type="int32"/>
        </struct>
        <struct name= "MyType">
            <member name="my_long" type="int32"/>
            <member name="my_string" stringMaxLength="512" type="string"/>
            <member name="my_foo" type="nonBasic" nonBasicTypeName= "Foo"/>
            <member name="my_sequence" sequenceMaxLength="10" type="int32"/>
            <member name="my_array" type="nonBasic" nonBasicTypeName= "Foo" arrayDimensions="5"/>
            <member name="my_optional" type="nonBasic" nonBasicTypeName= "Foo" optional="true"/>
        </struct>
    </types>
</dds>
```

And then load it as follows:

```
    using namespace dds::core::xtypes;
    // Create a QosProvider (or use the default one)
    dds::core::QosProvider qos_provider("path/to/MyType.xml");
    // Get the type called MyType
    const DynamicType& mytype = qos_provider.extensions().type("MyType");
    // Downcast to StructType if needed
    const StructType& mytype_struct = static_cast<const StructType&>(mytype);
    rti::core::xtypes::print_idl(mytype);
```

See also

Qos Provider Use Cases (p. 604)

6.214.1.3 Get the DynamicType from its equivalent IDL-generated type

<<***extension***>> *(p. 354)* When we have the IDL type definition at hand, obtaining its equivalent DynamicType is as simple as:

```
    const dds::core::xtypes::StructType& mytype = rti::topic::dynamic_type<MyType>::get();
    rti::core::xtypes::print_idl(mytype);
```

6.214.1.4 Create a DynamicType using tuples

<<***C++11***>> *(p. 353)* <<***experimental***>> *(p. 355)* <<***extension***>> *(p. 354)* Note this code requires the additional header `<rti/core/xtypes/DynamicDataTuples.hpp>`.

Since this feature only supports structures containing primitive types or strings, we'll create the following simpler IDL type:

```
struct MyOtherType {
    long m1;
    double m2;
    string m3;
};
```

We can create the type in a single line:

```
    dds::core::xtypes::StructType mytype = rti::core::xtypes::create_type_from_tuple<
            int32_t, double, std::string>("MyOtherType");
    rti::core::xtypes::print_idl(mytype);
```

For convenience, we can also use a `std::tuple`, which allows us to typedef it:

```
    typedef std::tuple<int32_t, double, std::string> MyOtherTypeTuple;
    mytype = rti::core::xtypes::create_type_from_tuple<MyOtherTypeTuple>("MyOtherType");
    rti::core::xtypes::print_idl(mytype);
```

6.214.2 Using DynamicData

Now that we know how to create a DynamicType, we will see how to use DynamicData to publish and subscribe to data that we can manipulate reflectively.

We will use the DynamicType that we **created before** (p. 609).

6.214.2.1 Publishing data using DynamicData

Make sure you are already familiar with the regular **publication example** (p. 304).

Unlike the regular publication example, using dynamic data we can write code where the type is unknown at compilation type.

```
using namespace dds::core::xtypes;
StructType mytype = create_mytype(); // create the type as we saw before
dds::domain::DomainParticipant participant(0);
// The template parameter is DynamicData. The third argument is the
// DynamicType, instead of the type name.
dds::topic::Topic<DynamicData> topic(participant, "MyTopic", mytype);
dds::pub::DataWriter<DynamicData> writer(dds::pub::Publisher(participant), topic);
// Create a data sample and assign its values
DynamicData sample(mytype);
sample.value("my_long", 23); // my_long = 23
sample.value<std::string>("my_string", "hello"); // my_string = hello
// To set my_foo we have two options:
// 1) Create a new DynamicData and assigning it:
// We can get the DynamicType of Foo from MyType::my_foo
DynamicData foo_data(mytype.member("my_foo").type());
foo_data.value("x", 1);
foo_data.value("y", 2);
sample.value("my_foo", foo_data);
// 2) Obtain the loaned member and assign its values
rti::core::xtypes::LoanedDynamicData loaned_member = sample.loan_value("my_foo");
loaned_member.get().value("x", 2);
loaned_member.get().value("y", 3);
loaned_member.return_loan(); // The destructor would do this as well.
// To set the values of my_sequence we can use a vector
std::vector<int32_t> sequence_values;
sequence_values.push_back(10);
sequence_values.push_back(20);
sequence_values.push_back(30);
sample.set_values("my_sequence", sequence_values);
// To set the values of my_array, since the element type is another struct
// we will use a LoanedDynamicData to access the array elements. To modify
// each element we will use another LoanedDynamicData
loaned_member = sample.loan_value("my_array");
rti::core::xtypes::LoanedDynamicData array_element = loaned_member.get().loan_value(1);
array_element.get().value("x", 10);
array_element.get().value("y", 11);
loaned_member.get().loan_value(array_element, 2); // reuse array_element
array_element.get().value("x", 20);
array_element.get().value("y", 21);
array_element.return_loan();
loaned_member.return_loan();
// We're not setting the remaining 3 elements, so they will have default values
// We manipulate optional members as regular members. By setting a value
// we are implicitly asserting its presence in the sample.
sample.value("my_optional", foo_data);
// This is how we can unset it:
sample.clear_optional_member("my_optional");
// Finally we just write the sample
writer.write(sample);
std::cout « sample « std::endl;
```

6.214.2.2 Subscribing to data using DynamicData

```
using namespace dds::core::xtypes;
using namespace rti::core::xtypes;
using namespace std; // for cout and endl
StructType mytype = create_mytype(); // create the type as we saw before
dds::domain::DomainParticipant participant(0);
// The template parameter is DynamicData. The third argument is the
// DynamicType, instead of the type name.
dds::topic::Topic<DynamicData> topic(participant, "MyTopic", mytype);
dds::sub::DataReader<DynamicData> reader(dds::sub::Subscriber(participant), topic);
// ...
// Read/take samples normally
dds::sub::LoanedSamples<DynamicData> samples = reader.take();
for (auto sample : samples) {
    if (sample.info().valid()) {
        DynamicData& data = const_cast<DynamicData&>(sample.data());
        cout « data.value<int32_t>("my_long") « endl;
        // To inspect a complex member we can get a copy of it:
        DynamicData foo_copy = data.value<DynamicData>("my_foo");
        cout « foo_copy.value<int32_t>("x") « endl;
        // Or a loan:
        LoanedDynamicData loaned_member = data.loan_value("my_foo");
        cout « loaned_member.get().value<int32_t>("y") « endl;
        loaned_member.return_loan(); // Note: the destructor also returns the loan if needed
        // Note that we could have taken advantage of the destructor to save
        // the call to return_loan:
        cout « data.loan_value("my_foo").get().value<int32_t>("y") « endl;
        // To get the values of my_sequence we can use a vector:
        std::vector<int32_t> sequence_values = data.get_values<int32_t>("my_sequence");
        // We could also have reused a vector:
        data.get_values("my_sequence", sequence_values);
        // To get the values of my_array, since the element type is another
        // struct we will use a LoanedDynamicData to access the array
        // elements.
        loaned_member = data.loan_value("my_array"); // Note: this move-assignment only works in C++11
        for (size_t i = 1; i <= loaned_member.get().member_count(); i++) {
            LoanedDynamicData array_element = loaned_member.get().loan_value(i);
            cout « array_element.get().value<int32_t>("x") « endl;
            cout « array_element.get().value<int32_t>("y") « endl;
        }
        // When getting an unset optional member, value() will throw
        // dds::core::InvalidArgumentError, but we can check if it's set:
        if (data.member_exists("my_optional")) {
            foo_copy = data.value<DynamicData>("my_optional");
            cout « foo_copy.value<int32_t>("x") « endl;
            cout « foo_copy.value<int32_t>("y") « endl;
        } else {
            cout « "my_optional is not set" « endl;
        }
    }
}
```

6.214.2.3 Manipulating data reflectively using C++ tuples

*<<**C++11**>> (p. 353) <<**experimental**>> (p. 355) <<**extension**>> (p. 354)*

This feature allows publishing and subscribing to simple data types without code generation and directly using C++ types instead of the DynamicData getters and setters.

This is an example publisher:

```
using namespace dds::core::xtypes;
// Create the type
StructType mytype = rti::core::xtypes::create_type_from_tuple<
    int32_t, double, std::string>("MyType");
// Create participant, topic and writer
dds::domain::DomainParticipant participant(0);
dds::topic::Topic<DynamicData> topic(participant, "MyTopic", mytype);
dds::pub::DataWriter<DynamicData> writer(dds::pub::Publisher(participant), topic);
// Create a data sample and assign its values
DynamicData sample(mytype);
rti::core::xtypes::set_tuple(
    sample, std::make_tuple(10, 50.5, std::string("Hello")));
// Write the sample
```

```
writer.write(sample);
std::cout « sample « std::endl;
```

And this is a subscriber:

```
using namespace dds::core::xtypes;
using namespace rti::core::xtypes;
// Create the type
StructType mytype = rti::core::xtypes::create_type_from_tuple<
    int32_t, double, std::string>("MyType");
dds::domain::DomainParticipant participant(0);
dds::topic::Topic<DynamicData> topic(participant, "MyTopic", mytype);
dds::sub::DataReader<DynamicData> reader(dds::sub::Subscriber(participant), topic);
// Read/take samples normally
dds::sub::LoanedSamples<DynamicData> samples = reader.take();
for (auto sample : samples) {
    if (sample.info().valid()) {
        // We obtain a std::tuple from the DynamicData sample
        auto my_tuple =  rti::core::xtypes::get_tuple<
            int32_t, double, std::string>(sample.data());
        std::cout « std::get<0>(my_tuple) « std::endl;
        std::cout « std::get<1>(my_tuple) « std::endl;
        std::cout « std::get<2>(my_tuple) « std::endl;
    }
}
```

Chapter 7

Namespace Documentation

7.1 dds Namespace Reference

The dds namespace. The dds namespace includes all of the standard types, classes, and functions.

Namespaces

- namespace **all**

  <<***extension***>> *(*p. *354) A single namespace where all standard symbols are included.*
- namespace **core**

  The core namespace contains infrastructure types and functions.
- namespace **domain**

  The domain namespace contains types, classes, and functions related to DomainParticipants.
- namespace **pub**

  Contains the type and functions to support publishing topics.
- namespace **sub**

  Contains the types and functions to support subscribing to topics.
- namespace **topic**

  Contains topic related classes and functions, the built-in topics and topic traits used by IDL-generated types.

7.1.1 Detailed Description

The dds namespace. The dds namespace includes all of the standard types, classes, and functions.

7.2 dds::all Namespace Reference

<<***extension***>> *(*p. *354)* A single namespace where all standard symbols are included.

7.2.1 Detailed Description

<<***extension***>> *(*p. *354)* A single namespace where all standard symbols are included.

The namespace **dds::all** (p. 615) brings all symbols to a single namespace as an alternative way of using the different namespaces such as **dds::core** (p. 616), **dds::core::cond** (p. 627), **dds::sub** (p. 658), etc.

For example:

```
#include <dds/dds.hpp>
void test_namespace()
{
    using namespace dds::all;
    DomainParticipant participant(MY_DOMAIN_ID); // dds::domain
    Topic<StringTopicType> topic(participant, "My Topic"); // dds::topic and dds::core
}
```

7.3 dds::core Namespace Reference

The core namespace contains infrastructure types and functions.

Namespaces

- namespace **cond**

  Contains the ***Condition*** *(*p. *948) classes.*
- namespace **policy**

  Contains the standard Qos policy classes.
- namespace **status**

  Contains the Status and State classes.
- namespace **xtypes**

  Contains the types and functions to support Extensible Types.

Classes

- class **AlreadyClosedError**

  Indicates that an object has been closed.
- class **basic_string**

  <<***value-type***>> *(*p. *350) A string convertible to std::string and with similar functionality*
- class **BytesTopicType**

  Built-in type consisting of a variable-length array of opaque bytes.
- class **Duration**

  <<***value-type***>> *(*p. *350) Represents a time interval*
- class **Entity**

  <<***reference-type***>> *(*p. *351) This is the abstract base class for all the DDS objects that support QoS policies, a listener and a status condition.*
- class **Error**

  A generic, unspecified ***Error*** *(*p. *1539).*
- class **Exception**

The abstract base class for all of the DDS exceptions which may be thrown by the API.

- class **external**

  A managed reference to an object.

- class **IllegalOperationError**

  Indicates that an operation was called under improper circumstances.

- class **ImmutablePolicyError**

  Indicates that the application attempted to modify an immutable QoS policy.

- class **InconsistentPolicyError**

  Indicates that the application specified a set of QoS policies that are not consistent with each other.

- class **InstanceHandle**

  *<<**value-type**>> (p. 350) Handle to identify different instances of the same **dds::topic::Topic** (p. 2474) of a certain type.*

- class **InvalidArgumentError**

  Indicates that the application passed an illegal parameter value into an operation.

- class **InvalidDowncastError**

  Indicates that a downcast was incorrect.

- class **KeyedBytesTopicType**

  Built-in type consisting of a variable-length array of opaque bytes and a string that is the key.

- class **KeyedStringTopicType**

  Built-in type consisting of a string payload and a second string that is the key.

- class **NotAllowedBySecurityError**

  Indicates that an operation on the DDS API fails because the security plugins do not allow it.

- class **NotEnabledError**

  *A **NotEnabledError** (p. 1869) is thrown when an operation is invoked on a **dds::core::Entity** (p. 1520) that is not yet enabled.*

- class **NullReferenceError**

  Indicates an attempt to access a null object.

- class **optional**

  *<<**value-type**>> (p. 350) Represents an object that may not contain a valid value*

- class **OutOfResourcesError**

  Indicates that RTI Connext ran out of the resources needed to complete the operation.

- class **PreconditionNotMetError**

  *A **PreconditionNotMetError** (p. 1941) is thrown when a pre-condition for the operation was not met.*

- class **QosProvider**

  *<<**reference-type**>> (p. 351) The **QosProvider** (p. 2025) class provides a way for a user to control and access the XML QoS profiles that are loaded by RTI Connext*

- class **Reference**

  Base class for all reference types.

- class **safe_enum**

  *<<**value-type**>> (p. 350) Provides a safe, scoped enumeration based on def::type*

- class **StringTopicType**

  Built-in type consisting of a single character string.

- class **TEntityQos**

  Acts as a container for Qos policies allowing to set and retrieve all the policies of an entity as a unit.

- class **Time**

  *<<**extension**>> (p. 354) Represents a point in time*

- class **TimeoutError**

  Indicates that an operation has timed out.

- class **UnsupportedError**

  Indicates that the application used an unsupported operation.
- class **Value**
- class **vector**

  *<<**value-type**>> (p. 350) A vector convertible to std::vector and with similar functionality*
- class **WeakReference**

Typedefs

- typedef std::vector< **InstanceHandle** > **InstanceHandleSeq**

  *A sequence of **dds::core::InstanceHandle** (p. 1612).*
- typedef **basic_string**< char, rti::core::memory::OsapiAllocator< char > > **string**

  A string convertible to std::string and with similar functionality.
- typedef **basic_string**< DDS_Wchar, rti::core::memory::OsapiAllocator< DDS_Wchar > > **wstring**

  An IDL-derived wide string.
- typedef std::vector< uint8_t > **ByteSeq**

  A vector of bytes.
- typedef std::vector< std::string > **StringSeq**

  A vector of strings.
- typedef std::nullptr_t **null_type**

  *<<**C++11**>> (p. 353) The type of **dds::core::null** (p. 438)*

Functions

- template<typename TO , typename FROM >
  TO **polymorphic_cast** (const FROM &from)

  *Downcasts an **Entity** (p. 1520) to a subclass.*
- **Duration operator*** (uint32_t lhs, const **Duration** &rhs)

  *Multiply a **Duration** (p. 1448) object by an unsigned integer.*
- **Duration operator*** (const **Duration** &lhs, uint32_t rhs)

  *Multiply a **Duration** (p. 1448) object by an unsigned integer.*
- **Duration operator/** (const **Duration** &lhs, uint32_t rhs)

  *Divide a **Duration** (p. 1448) object by an unsigned integer.*
- void **swap** (**Duration** &lhs, **Duration** &rhs) OMG_NOEXCEPT

  *Swap the contents of two **Duration** (p. 1448) objects.*
- template<typename def , typename inner >
  std::ostream & **operator<<** (std::ostream &out, const **safe_enum**< def, inner > &the_enum)

  Applies `operator<<` to the underlying enum.
- template<typename CharType , typename Allocator >
  std::ostream & **operator<<** (std::ostream &out, const **basic_string**< CharType, Allocator > &the_string)

  Prints the string.
- **Time operator+** (const **Time** &time, const **Duration** &duration)

  *Add a **Time** (p. 2457) and a **Duration** (p. 1448) together.*
- **Time operator+** (const **Duration** &duration, const **Time** &time)

  *Add a **Time** (p. 2457) and a **Duration** (p. 1448) together.*
- **Time operator-** (const **Time** &time, const **Duration** &duration)

*Subtract a **Duration** (p. 1448) from a **Time** (p. 2457).*

- **Duration operator-** (const **Time** &time1, const **Time** &time2)

  Calculate the duration between two times.

- template<typename T >
  bool **operator==** (const **optional**< T > &a, const **optional**< T > &b)

  Compares two optional values.

- template<typename T >
  bool **operator!=** (const **optional**< T > &a, const **optional**< T > &b)

  Compares two optional values.

- template<typename T >
  bool **operator==** (const **optional**< T > &optional_value, const T &value)

  Compares an optional member and a value of the underlying type.

- template<typename T >
  bool **operator==** (const T &value, const **optional**< T > &optional_value)

  Compares an optional member and a value of the underlying type.

- template<typename T >
  bool **operator!=** (const **optional**< T > &optional_value, const T &value)

  Compares an optional member and a value of the underlying type.

- template<typename T >
  bool **operator!=** (const T &value, const **optional**< T > &optional_value)

  Compares an optional member and a value of the underlying type.

- template<typename T >
  std::ostream & **operator**<< (std::ostream &out, const **optional**< T > & **optional**)

  Applies `operator<<` to `*optional` or to the string `"NULL"` if `!optional.has_value()`.

- template<typename T >
  std::ostream & **operator**<< (std::ostream &out, const **vector**< T > &v)

  Print a vector applying << to all of its elements.

Variables

- const int32_t **LENGTH_UNLIMITED** = -1

  A special value indicating an unlimited quantity.

- const **null_type null**

  Indicates an empty reference.

7.3.1 Detailed Description

The core namespace contains infrastructure types and functions.

7.3.2 Typedef Documentation

7.3.2.1 InstanceHandleSeq

```
typedef std::vector< InstanceHandle>  dds::core::InstanceHandleSeq
```

A sequence of **dds::core::InstanceHandle** (p. 1612).

7.3.3 Function Documentation

7.3.3.1 polymorphic_cast()

```
template<typename TO , typename FROM >
TO polymorphic_cast (
             const FROM & from )
```

Downcasts an **Entity** (p. 1520) to a subclass.

Note

- Header: <dds/core/ref_traits.hpp>
- Namespace: **dds::core** (p. 616)

Template Parameters

TO	The type to cast to
FROM	The type to cast from

Parameters

from	The object to cast to type TO

Returns

A reference to the same object from, but cast to the type TO. Both objects share ownership of the underlying entity.

For example:

```
using namespace dds::core;
using namespace dds::domain;
DomainParticipant participant(0);
Entity entity = participant; // Assignment to base
DomainParticipant participant2 = polymorphic_cast<DomainParticipant>(entity);
assert (participant == participant2); // Same reference
```

See also

Reference types (p. 351)

7.3.3.2 operator*() [1/2]

```
Duration operator* (
            uint32_t lhs,
            const  Duration & rhs )
```

Multiply a **Duration** (p. 1448) object by an unsigned integer.

Parameters

lhs	The unsigned integer to multiply the **Duration** (p. 1448) object by
rhs	The **Duration** (p. 1448) object to multiply

Returns

Duration (p. 1448) The result of the multiplication

7.3.3.3 operator*() [2/2]

```
Duration operator* (
            const  Duration & lhs,
            uint32_t rhs )
```

Multiply a **Duration** (p. 1448) object by an unsigned integer.

Parameters

lhs	The **Duration** (p. 1448) object to multiply
rhs	The unsigned integer to multiply the **Duration** (p. 1448) object by

Returns

Duration (p. 1448) The result of the multiplication

7.3.3.4 operator/()

```
Duration operator/ (
            const  Duration & lhs,
            uint32_t rhs )
```

Divide a **Duration** (p. 1448) object by an unsigned integer.

Parameters

lhs	The dividend
rhs	The divisor

Returns

Duration (p. 1448) The result of dividing the two **Duration** (p. 1448) objects

7.3.3.5 swap()

```
void swap (
              Duration & lhs,
              Duration & rhs )
```

Swap the contents of two **Duration** (p. 1448) objects.

Parameters

lhs	One of the **Duration** (p. 1448) objects
rhs	One of the **Duration** (p. 1448) objects

7.3.3.6 operator<<() [1/4]

```
template<typename def , typename inner >
std::ostream & operator<< (
             std::ostream & out,
             const  safe_enum< def, inner > & the_enum )
```

Applies operator<< to the underlying enum.

7.3.3.7 operator<<() [2/4]

```
template<typename CharType , typename Allocator >
std::ostream & operator<< (
             std::ostream & out,
             const  basic_string< CharType, Allocator > & the_string )
```

Prints the string.

7.3.3.8 operator+() [1/2]

```
Time operator+ (
            const  Time & time,
            const  Duration & duration )
```

Add a **Time** (p. 2457) and a **Duration** (p. 1448) together.

Parameters

time	The **Time** (p. 2457) object to add
duration	The **Duration** (p. 1448) object to add

Returns

Time (p. 2457) The result of the addition represented as a **Time** (p. 2457) object

7.3.3.9 operator+() [2/2]

```
Time operator+ (
            const  Duration & duration,
            const  Time & time )
```

Add a **Time** (p. 2457) and a **Duration** (p. 1448) together.

Parameters

duration	The **Duration** (p. 1448) object to add
time	The **Time** (p. 2457) object to add

Returns

Time (p. 2457) The result of the addition represented as a **Time** (p. 2457) object

7.3.3.10 operator-() [1/2]

```
Time operator- (
            const  Time & time,
            const  Duration & duration )
```

Subtract a **Duration** (p. 1448) from a **Time** (p. 2457).

Parameters

time	The **Time** (p. 2457) object to subtract from
duration	The **Duration** (p. 1448) object to subtract

Returns

Time (p. 2457) The result of the subtraction represented as a **Time** (p. 2457) object

7.3.3.11 operator-() [2/2]

```
Duration operator- (
            const  Time & time1,
            const  Time & time2 )
```

Calculate the duration between two times.

Parameters

time1	The first ("before") time
time2	The second ("after") time

Exceptions

std::overflow_error	if duration exceeds the infinite limit

Returns

The duration elapsed between time1 and time2. If time1 $>$ time2 the duration is zero. If time1 is **Time::maximum()** (p. 2459), the duration is **Duration::infinite()** (p. 1451);

7.3.3.12 operator==() [1/3]

```
template<typename T >
bool operator== (
            const  optional< T > & a,
            const  optional< T > & b )
```

Compares two optional values.

Returns

true if both are unset or both are set and `*a == *b`.

7.3.3.13 operator"!=() [1/3]

```
template<typename T >
bool operator!= (
            const  optional< T > & a,
            const  optional< T > & b )
```

Compares two optional values.

Returns

false if both are unset or both are set and `*a == *b`.

7.3.3.14 operator==() [2/3]

```
template<typename T >
bool operator== (
            const  optional< T > & optional_value,
            const T & value )
```

Compares an optional member and a value of the underlying type.

Returns

Return true if `optional_value` is set and `*optional_value == value`

7.3.3.15 operator==() [3/3]

```
template<typename T >
bool operator== (
            const T & value,
            const  optional< T > & optional_value )
```

Compares an optional member and a value of the underlying type.

Returns

Return true if optional_value is set and `*optional_value == value`

7.3.3.16 operator"!=() [2/3]

```
template<typename T >
bool operator!= (
            const  optional< T > & optional_value,
            const T & value )
```

Compares an optional member and a value of the underlying type.

Returns

Return false if optional_value is set and `*optional_value == value`

7.3.3.17 operator"!=() [3/3]

```
template<typename T >
bool operator!= (
            const T & value,
            const  optional< T > & optional_value )
```

Compares an optional member and a value of the underlying type.

Returns

Return false if optional_value is set and `*optional_value == value`

7.3.3.18 operator<<() [3/4]

```
template<typename T >
std::ostream & operator<< (
            std::ostream & out,
            const  optional< T > & optional )
```

Applies `operator<<` to `*optional` or to the string `"NULL"` if `!optional.has_value()`.

7.3.3.19 operator<<() [4/4]

```
template<typename T >
std::ostream & operator<< (
            std::ostream & out,
            const  vector< T > & v )
```

Print a vector applying << to all of its elements.

7.4 dds::core::cond Namespace Reference

Contains the **Condition** (p. 948) classes.

Classes

- class **Condition**

  <<***reference-type***>> *(p. 351) Abstract base class of all the conditions*
- class **GuardCondition**

  <<***reference-type***>> *(p. 351) A condition whose trigger value is under the control of the application.*
- class **StatusCondition**

  <<***reference-type***>> *(p. 351) A condition associated with each* ***dds::core::Entity*** *(p. 1520)*
- class **WaitSet**

  <<***reference-type***>> *(p. 351) Allows an application to wait until one or more of the attached* ***Condition*** *(p. 948) objects have a trigger_value of true or else until the timeout expires.*

7.4.1 Detailed Description

Contains the **Condition** (p. 948) classes.

7.5 dds::core::policy Namespace Reference

Contains the standard Qos policy classes.

Classes

- class **DataRepresentation**

  Contains the data representations supported by entities.
- class **DataTag**

  Stores name-value (string) pairs that can be used to determine access permissions.
- class **Deadline**

  Expresses the maximum duration (deadline) within which an instance is expected to be updated.
- class **DestinationOrder**

  Controls the logical order of updates to the same instance by a ***dds::pub::Publisher*** *(p. 1993).*
- struct **DestinationOrderKind_def**

  The definition of the ***dds::core::safe_enum*** *(p. 2265) DestinationOrderKind.*
- class **Durability**

  Specifies whether a ***dds::pub::DataWriter*** *(p. 1138) will store and deliver previously published data samples to late-joining* ***dds::sub::DataReader*** *(p. 976) entities.*
- struct **DurabilityKind_def**

  The definition of the ***dds::core::safe_enum*** *(p. 2265) DurabilityKind.*
- class **DurabilityService**

  Configures the external RTI Persistence Service used by persistent and transient DataWriters.

- class **EntityFactory**

  *Configures a **dds::core::Entity*** (p. *1520*) *that acts as factory of other entities.*
- class **GroupData**
- class **History**

  *Specifies how much historical data a **dds::pub::DataWriter*** (p. *1138*) *and a **dds::sub::DataReader*** (p. *976*) *can store.*
- struct **HistoryKind_def**

  *The definition of the **dds::core::safe_enum*** (p. *2265*) *HistoryKind.*
- class **LatencyBudget**

  Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.
- class **Lifespan**

  *Specifies how long the data written by a **dds::pub::DataWriter*** (p. *1138*) *is considered valid.*
- class **Liveliness**

  *Specifies and configures the mechanism that allows **dds::sub::DataReader*** (p. *976*)*'s to detect when **dds::pub::Data↩ Writer*** (p. *1138*)*'s become disconnected.*
- struct **LivelinessKind_def**

  *The definition of the **dds::core::safe_enum*** (p. *2265*) *LivelinessKind.*
- class **Ownership**

  *Specifies whether it is allowed for multiple **dds::pub::DataWriter*** (p. *1138*)*'s to write the same instance of the data and if so, how these modifications should be arbitrated.*
- struct **OwnershipKind_def**

  *The definition of the **dds::core::safe_enum*** (p. *2265*) *OwnershipKind.*
- class **OwnershipStrength**

  *Specifies the value of the strength used to arbitrate among multiple **dds::pub::DataWriter*** (p. *1138*) *objects that attempt to modify the same instance of a data type (identified by its **dds::topic::Topic*** (p. *2474*) *and key).*
- class **Partition**

  *Set of strings that introduces logical partitions in **dds::domain::DomainParticipant*** (p. *1326*)*, **dds::pub::Publisher*** (p. *1993*)*, or **dds::sub::Subscriber*** (p. *2410*) *entities.*
- class **policy_id**

  Obtains the QosPolicyId of a QoS Policy.
- class **policy_name**

  Obtains the policy name.
- class **Presentation**

  Specifies how the samples representing changes to data instances are presented to a subscribing application.
- struct **PresentationAccessScopeKind_def**

  *The definition of the **dds::core::safe_enum*** (p. *2265*) *PresentationAccessScopeKind.*
- class **QosPolicyCount**

  *<<**value-type**>>* (p. *350*) *Holds a counter for a QosPolicyId*
- class **ReaderDataLifecycle**

  Controls how a DataReader manages the lifecycle of the data that it has received.
- class **Reliability**

  *Indicates the level of reliability in sample delivered that a **dds::pub::DataWriter*** (p. *1138*) *offers or a **dds::sub::Data↩ Reader*** (p. *976*) *requests.*
- struct **ReliabilityKind_def**

  *The definition of the **dds::core::safe_enum*** (p. *2265*) *ReliabilityKind.*
- class **ResourceLimits**

  *Controls the memory usage of **dds::pub::DataWriter*** (p. *1138*) *or a **dds::sub::DataReader*** (p. *976*)*.*
- class **TimeBasedFilter**

*Allows a **dds::sub::DataReader** (p. 976) to indicate that it is not interested in all the sample updates that occur within a time period.*

- class **TopicData**
- class **TransportPriority**

  Allows applications to take advantage of transports capable of sending messages with different priorities.
- class **TypeConsistencyEnforcement**

  Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.
- struct **TypeConsistencyEnforcementKind_def**

  *The definition of the **dds::core::safe_enum** (p. 2265) TypeConsistencyEnforcementKind.*
- class **UserData**

  *Attaches a buffer of opaque data that is distributed by **Built-in Topics** (p. 46) during discovery.*
- class **WriterDataLifecycle**

  *Controls how a **dds::pub::DataWriter** (p. 1138) handles the lifecycle of the instances (keys) that it writes.*

Typedefs

- typedef **dds::core::safe_enum**< **OwnershipKind_def** > **OwnershipKind**

  ***Safe Enumeration** (p. 429) of **OwnershipKind_def** (p. 1905)*
- typedef **dds::core::safe_enum**< **DurabilityKind_def** > **DurabilityKind**

  ***Safe Enumeration** (p. 429) of **DurabilityKind_def** (p. 1441)*
- typedef **dds::core::safe_enum**< **PresentationAccessScopeKind_def** > **PresentationAccessScopeKind**

  ***Safe Enumeration** (p. 429) of **PresentationAccessScopeKind_def** (p. 1950)*
- typedef **dds::core::safe_enum**< **ReliabilityKind_def** > **ReliabilityKind**

  ***Safe Enumeration** (p. 429) of **ReliabilityKind_def** (p. 2156)*
- typedef **dds::core::safe_enum**< **DestinationOrderKind_def** > **DestinationOrderKind**

  ***Safe Enumeration** (p. 429) of **DestinationOrderKind_def** (p. 1263)*
- typedef **dds::core::safe_enum**< **HistoryKind_def** > **HistoryKind**

  ***Safe Enumeration** (p. 429) of **HistoryKind_def** (p. 1606)*
- typedef **dds::core::safe_enum**< **LivelinessKind_def** > **LivelinessKind**

  ***Safe Enumeration** (p. 429) of **LivelinessKind_def** (p. 1654)*
- typedef **dds::core::safe_enum**< **TypeConsistencyEnforcementKind_def** > **TypeConsistency↩ EnforcementKind**

  ***Safe Enumeration** (p. 429) of **TypeConsistencyEnforcementKind_def** (p. 2570)*
- typedef std::vector< **QosPolicyCount** > **QosPolicyCountSeq**

  *A vector of **QosPolicyCount** (p. 2020).*
- typedef int16_t **DataRepresentationId**

  *The type of the elements that **DataRepresentation** (p. 1113) contains.*
- typedef std::vector< **DataRepresentationId** > **DataRepresentationIdSeq**

  A vector of DataRepresentationId.
- typedef uint32_t **QosPolicyId**

  Identifies a QoS policy.

7.5.1 Detailed Description

Contains the standard Qos policy classes.

7.6 dds::core::status Namespace Reference

Contains the Status and State classes.

Classes

- class **InconsistentTopicStatus**

  *Information about the status **dds::core::status::StatusMask::inconsistent_topic()** (p. 2379)*
- class **LivelinessChangedStatus**

  *Information about the status **dds::core::status::StatusMask::liveliness_changed()** (p. 2383)*
- class **LivelinessLostStatus**

  *Information about the status **dds::core::status::StatusMask::liveliness_lost()** (p. 2383)*
- class **OfferedDeadlineMissedStatus**

  *Information about the status **dds::core::status::StatusMask::offered_deadline_missed()** (p. 2379)*
- class **OfferedIncompatibleQosStatus**

  *Information about the status **dds::core::status::StatusMask::offered_incompatible_qos()** (p. 2380)*
- class **PublicationMatchedStatus**

  *Information about the status **dds::core::status::StatusMask::publication_matched()** (p. 2384)*
- class **RequestedDeadlineMissedStatus**

  *Information about the status **dds::core::status::StatusMask::requested_deadline_missed()** (p. 2380)*
- class **RequestedIncompatibleQosStatus**

  *Information about the status **dds::core::status::StatusMask::requested_incompatible_qos()** (p. 2381)*
- class **SampleLostStatus**

  *Information about the status **dds::core::status::StatusMask::sample_lost()** (p. 2381)*
- class **SampleRejectedState**

  Reasons why a sample was rejected.
- class **SampleRejectedStatus**

  *Information about the status **dds::core::status::StatusMask::sample_rejected()** (p. 2382)*
- class **StatusMask**

  A std::bitset (list) of statuses.
- class **SubscriptionMatchedStatus**

  *Information about the status **dds::core::status::StatusMask::subscription_matched()** (p. 2384)*

Functions

- template<typename STATUS >
  StatusMask get_status ()

  *Obtains the **StatusMask** (p. 2375) mask associated to a status class.*

7.6.1 Detailed Description

Contains the Status and State classes.

7.6.2 Function Documentation

7.6.2.1 get_status()

```
template<typename STATUS >
StatusMask dds::core::status::get_status ( )
```

Obtains the **StatusMask** (p. 2375) mask associated to a status class.

For example:
```
using namespace dds::core::status;
StatusMask mask = get_status<PublicationMatchedStatus>();
assert(mask == StatusMask::publication_matched());
```

7.7 dds::core::xtypes Namespace Reference

Contains the types and functions to support Extensible Types.

Classes

- class **AbstractConstructedType**

  The base class of types that have members and an extensibility kind.
- class **AliasType**

  <<***value-type***>> *(p. 350) Represents an IDL* `typedef`
- class **ArrayType**

  <<***value-type***>> *(p. 350) Represents an IDL array type.*
- class **CollectionType**

  <<***value-type***>> *(p. 350) The base class of all collection types*
- class **DynamicData**

  <<***value-type***>> *(p. 350) A data sample of any complex data type, which can be inspected and manipulated reflectively.*
- class **DynamicType**

  <<***value-type***>> *(p. 350) Represents a runtime type.*
- class **EnumMember**

  <<***value-type***>> *(p. 350) Represents a* ***EnumType*** *(p. 1535) member*
- class **EnumType**

  <<***value-type***>> *(p. 350) Represents an IDL* `enum` *type*
- struct **ExtensibilityKind_def**

  The definition of the ***dds::core::safe_enum*** *(p. 2265) ExtensibilityKind.*
- class **Member**

  <<***value-type***>> *(p. 350) Represents a* ***StructType*** *(p. 2400) member*
- class **PrimitiveType**

  <<***value-type***>> *(p. 350) Represents an IDL primitive type*
- class **SequenceType**

<<***value-type***>> (p. *350) Represents an IDL* `sequence` *type.*

- class **StringType**

  <<***value-type***>> (p. *350) Represents an IDL* `string` *type.*
- class **StructType**

  <<***value-type***>> (p. *350) Represents an IDL* `struct` *type*
- struct **TypeKind_def**

  The definition of TypeKind.
- class **UnidimensionalCollectionType**

  <<***value-type***>> (p. *350) The base class of collection types with only one dimension.*
- class **UnionMember**

  <<***value-type***>> (p. *350) Represents a* ***UnionType*** (p. *2584) member*
- class **UnionType**

  <<***value-type***>> (p. *350) Represents an IDL* `union` *type*
- class **WStringType**

  <<***value-type***>> (p. *350) Represents an IDL* `wstring` *type.*

Typedefs

- typedef **dds::core::safe_enum**< **ExtensibilityKind_def** > **ExtensibilityKind**

  The extensibility of a type.
- typedef **dds::core::safe_enum**< **TypeKind_def** > **TypeKind**

  The different type kinds.

Functions

- bool **is_primitive_type** (const **DynamicType** &t)

  Determines if a ***DynamicType*** (p. *1504) is a* ***PrimitiveType*** (p. *1959).*
- bool **is_constructed_type** (const **DynamicType** &t)

  Determines if a ***DynamicType*** (p. *1504) is a constructed type.*
- bool **is_collection_type** (const **DynamicType** &t)

  Determines if a ***DynamicType*** (p. *1504) is a* ***CollectionType*** (p. *939).*
- bool **is_aggregation_type** (const **DynamicType** &t)

  Determines if a ***DynamicType*** (p. *1504) is an aggregation type.*
- template<typename T >
  const **PrimitiveType**< T > & **primitive_type** ()

  Obtains a singleton of PrimitiveType<T>
- std::ostream & **operator**<< (std::ostream &out, const **DynamicType** &type)

  <<***extension***>> (p. *354) Converts the* ***DynamicType*** (p. *1504) to a string.*

7.7.1 Detailed Description

Contains the types and functions to support Extensible Types.

7.7.2 Typedef Documentation

7.7.2.1 ExtensibilityKind

```
typedef  dds::core::safe_enum< ExtensibilityKind_def>  dds::core::xtypes::ExtensibilityKind
```

The extensibility of a type.

See also

DynamicType (p. 1504)

7.7.3 Function Documentation

7.7.3.1 is_primitive_type()

```
bool is_primitive_type (
             const  DynamicType & t )
```

Determines if a **DynamicType** (p. 1504) is a **PrimitiveType** (p. 1959).

7.7.3.2 is_constructed_type()

```
bool is_constructed_type (
             const  DynamicType & t )
```

Determines if a **DynamicType** (p. 1504) is a constructed type.

This includes **EnumType** (p. 1535), **AliasType** (p. 801), collection types and aggregation types.

See also

is_collection_type (p. 633)

is_aggregation_type (p. 634)

7.7.3.3 is_collection_type()

```
bool is_collection_type (
            const  DynamicType & t )
```

Determines if a **DynamicType** (p. 1504) is a **CollectionType** (p. 939).

This includes **ArrayType** (p. 828), **SequenceType** (p. 2344), **StringType** (p. 2399), and **WStringType** (p. 2665).

7.7.3.4 is_aggregation_type()

```
bool is_aggregation_type (
            const  DynamicType & t )
```

Determines if a **DynamicType** (p. 1504) is an aggregation type.

This includes **StructType** (p. 2400) and **UnionType** (p. 2584).

7.7.3.5 primitive_type()

```
template<typename T >
const  PrimitiveType< T > & primitive_type ( )
```

Obtains a singleton of PrimitiveType<T>

7.7.3.6 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const  DynamicType & type )
```

<<***extension***>> *(p. 354)* Converts the **DynamicType** (p. 1504) to a string.

This operator writes the **DynamicType** (p. 1504) type to the ostream out using default values for **DynamicTypePrint↩ FormatProperty** (p. 1509) and returns a reference to out.

7.8 dds::domain Namespace Reference

The domain namespace contains types, classes, and functions related to DomainParticipants.

Namespaces

- namespace **qos**

  *Contains **DomainParticipantQos** (p. 1384).*

Classes

- class **DomainParticipant**

  *<<**reference-type**>> (p. 351) Container for all **dds::core::Entity** (p. 1520) objects.*
- class **DomainParticipantListener**

  *The listener class for a **DomainParticipant** (p. 1326).*
- class **NoOpDomainParticipantListener**

  *A convenience implementation of **DomainParticipantListener** (p. 1382) where all methods are overridden to do nothing.*

Functions

- void **ignore** (const **dds::domain::DomainParticipant** &participant, const **dds::core::InstanceHandle** &handle)

  *Instructs RTI Connext to locally ignore a remote **dds::domain::DomainParticipant** (p. 1326).*
- template<typename FwdIterator >
  void **ignore** (const **dds::domain::DomainParticipant** &participant, FwdIterator begin, FwdIterator end)

  *Instructs RTI Connext to locally ignore a group of remote **dds::domain::DomainParticipant** (p. 1326).*
- **dds::core::InstanceHandleSeq discovered_participants** (const **dds::domain::DomainParticipant** &participant)

  Retrieves the list of other participants discovered by this participant.
- template<typename FwdIterator >
  FwdIterator **discovered_participants** (const **dds::domain::DomainParticipant** &participant, FwdIterator begin, FwdIterator end)

  Retrieves the list of other participants discovered by this participant.
- **dds::topic::ParticipantBuiltinTopicData discovered_participant_data** (const **dds::domain::DomainParticipant** &participant, const **dds::core::InstanceHandle** &participant_handle)

  Retrieves the information about one participant discovered by this participant.
- **DomainParticipant find** (int32_t domain_id)

  *Locates an existing **dds::domain::DomainParticipant** (p. 1326).*

7.8.1 Detailed Description

The domain namespace contains types, classes, and functions related to DomainParticipants.

7.8.2 Function Documentation

7.8.2.1 ignore() [1/2]

```
void ignore (
            const  dds::domain::DomainParticipant & participant,
            const  dds::core::InstanceHandle & handle )
```

Instructs RTI Connext to locally ignore a remote **dds::domain::DomainParticipant** (p. 1326).

```
#include <dds/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **dds::domain** (p. 634)

From the time of this call onwards, RTI Connext will locally behave as if the remote participant did not exist. This means it will ignore any topic, publication, or subscription that originates on that **dds::domain::DomainParticipant** (p. 1326).

There is no way to reverse this operation.

This operation can be used in conjunction with the discovery of remote participants offered by means of the **dds::topic::ParticipantBuiltinTopicData** (p. 1908) to provide access control.

Application data can be associated with a **dds::domain::DomainParticipant** (p. 1326) by means of the **USER_DATA** (p. 554) policy. This application data is propagated as a field in the built-in topic and can be used by an application to implement its own access control policy.

The **dds::domain::DomainParticipant** (p. 1326) to ignore is identified by the handle argument. This `handle` is the one that appears in the **dds::sub::SampleInfo** (p. 2285) retrieved when reading the data-samples available for the built-in **dds::sub::DataReader** (p. 976) to the **dds::domain::DomainParticipant** (p. 1326) topic. The built-in **dds::sub::DataReader** (p. 976) is read with the same **dds::sub::DataReader::read** (p. 996) and **dds::sub::DataReader::take** (p. 997) operations used for any **dds::sub::DataReader** (p. 976).

Parameters

participant	The **DomainParticipant** (p. 1326) for which the remote entity will be ignored.
handle	The **dds::core::InstanceHandle** (p. 1612) of the remote entity that will be ignored.

Referenced by **dds::domain::DomainParticipant::ignore()**.

7.8.2.2 ignore() [2/2]

```
template<typename FwdIterator >
void ignore (
            const  dds::domain::DomainParticipant & participant,
            FwdIterator begin,
            FwdIterator end )
```

Instructs RTI Connext to locally ignore a group of remote **dds::domain::DomainParticipant** (p. 1326).

```
#include <dds/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **dds::domain** (p. 634)

The series of entities whose instance handles are made available via the provided iterators will be ignored.

Parameters

participant	The **DomainParticipant** (p. 1326) for which the remote entity will be ignored
begin	The begin iterator for the series of InstanceHandles to ignore
end	The end iterator for the series of InstanceHandles to ignore

See also

ignore(const dds::domain::DomainParticipant& participant, const dds::core::InstanceHandle& handle) (p. 635)

7.8.2.3 discovered_participants() [1/2]

```
dds::core::InstanceHandleSeq discovered_participants (
            const  dds::domain::DomainParticipant & participant )
```

Retrieves the list of other participants discovered by this participant.

```
#include <dds/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **dds::domain** (p. 634)

This operation retrieves the list of **dds::domain::DomainParticipant** (p. 1326) entities that have been discovered in the domain and that the application has not indicated should be "ignored" by means of the **dds::domain::ignore** (p. 635) operation. When using **rti::core::policy::DiscoveryConfigBuiltinPluginKindMask::SPDP2** (p. 1324), this list only includes **dds::domain::DomainParticipant** (p. 1326) entities that the application has received configuration information from.

Parameters

participant	The **DomainParticipant** (p. 1326) where to look up the discovered participants

Returns

The list of `InstanceHandles` that can be passed to **dds::domain::discovered_participant_data** (p. 638).

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869)

Referenced by **dds::domain::DomainParticipant::discovered_participants()**.

7.8.2.4 discovered_participants() [2/2]

```
template<typename FwdIterator >
FwdIterator discovered_participants (
            const  dds::domain::DomainParticipant & participant,
            FwdIterator begin,
            FwdIterator end )
```

Retrieves the list of other participants discovered by this participant.

```
#include <dds/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **dds::domain** (p. 634)

Template Parameters

FwdIterator	A forward iterator whose value type is **dds::core::InstanceHandleSeq** (p. 619)

This overload copies the instance handles into an iterator range.

Parameters

participant	The participant whose discovered participants are looked up
begin	The begin iterator where to copy the instance handles
end	One past the last position where instance handles are copied

7.8.2.5 discovered_participant_data()

```
dds::topic::ParticipantBuiltinTopicData discovered_participant_data (
            const  dds::domain::DomainParticipant & participant,
            const  dds::core::InstanceHandle & participant_handle )
```

Retrieves the information about one participant discovered by this participant.

```
#include <dds/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **dds::domain** (p. 634)

This operation retrieves information on a **dds::domain::DomainParticipant** (p. 1326) that has been discovered on the network. The participant must be in the same domain as the participant on which this operation is invoked and must not have been "ignored" by means of the **dds::domain::ignore** (p. 635) operation.

The `participant_handle` must correspond to such a **DomainParticipant** (p. 1326). Otherwise, the operation will fail with **dds::core::PreconditionNotMetError** (p. 1941).

Use the operation **dds::domain::discovered_participants** (p. 637) to find the **dds::domain::DomainParticipant** (p. 1326) entities that are currently discovered.

MT Safety:

Safe.

Parameters

participant	The **DomainParticipant** (p. 1326) where to look up the information
participant_handle	<<***in***>> *(p. 355)* **dds::core::InstanceHandle** (p. 1612) of **dds::domain::DomainParticipant** (p. 1326).

Returns

The participant information about `participant_handle`

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::PreconditionNotMetError** (p. 1941) or **dds::core::NotEnabledError** (p. 1869)

See also

dds::topic::ParticipantBuiltinTopicData (p. 1908)

dds::domain::discovered_participants (p. 637)

7.8.2.6 find()

```
DomainParticipant find (
            int32_t domain_id )
```

Locates an existing **dds::domain::DomainParticipant** (p. 1326).

```
#include <dds/domain/find.hpp>
```

Note

This is a standalone function in the namespace **dds::domain** (p. 634)

If no such **DomainParticipant** (p. 1326) exists, the operation will return **dds::core::null** (p. 438).

If multiple DomainParticipants belonging to that domain id exist, then the operation will return one of them. It is not specified which one.

Parameters

domain_id	The domain id of the **DomainParticipant** (p. 1326) to find

7.9 dds::domain::qos Namespace Reference

Contains **DomainParticipantQos** (p. 1384).

Classes

- class **DomainParticipantFactoryQos**

  <<***value-type***>> (p. *350) Container of the QoS policies that do not apply to a specific entity*

- class **DomainParticipantQos**

  <<***value-type***>> (p. *350) Container of the QoS policies that a **dds::domain::DomainParticipant** (p. 1326) supports*

Functions

- std::string **to_string** (const **DomainParticipantFactoryQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> (p. *354) Obtains a string representation of the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378)*

- std::string **to_string** (const **DomainParticipantFactoryQos** &qos, const **DomainParticipantFactoryQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> (p. *354) Obtains a string representation of the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378)*

- std::string **to_string** (const **DomainParticipantFactoryQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> (p. *354) Obtains a string representation of the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378)*

- std::ostream & **operator**<< (std::ostream &out, const rti::domain::qos::DomainParticipantFactoryQos &qos)

  <<***extension***>> (p. *354) Prints a dds::sub::qos::DomainParticipantFactoryQos to an output stream.*

- std::string **to_string** (const **DomainParticipantQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> (p. *354) Obtains a string representation of the **dds::domain::qos::DomainParticipantQos** (p. 1384)*

- std::string **to_string** (const **DomainParticipantQos** &qos, const **DomainParticipantQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::domain::qos::DomainParticipantQos*** *(p. 1384)*
- std::string **to_string** (const **DomainParticipantQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::domain::qos::DomainParticipantQos*** *(p. 1384)*
- std::ostream & **operator**<< (std::ostream &out, const rti::domain::qos::DomainParticipantQos &qos)

  <<***extension***>> *(p. 354) Prints a dds::sub::qos::DomainParticipantQos to an output stream.*

7.9.1 Detailed Description

Contains **DomainParticipantQos** (p. 1384).

7.9.2 Function Documentation

7.9.2.1 to_string() [1/6]

```
std::string to_string (
            const  DomainParticipantFactoryQos & qos,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
DomainParticipantFactoryQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for DomainParticipantFactoryQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
DomainParticipantFactoryQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
```

```
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

7.9.2.2 to_string() [2/6]

```
std::string to_string (
            const  DomainParticipantFactoryQos & qos,
            const  DomainParticipantFactoryQos & base,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

7.9.2.3 to_string() [3/6]

```
std::string to_string (
            const  DomainParticipantFactoryQos & qos,
            const  rti::core::qos_print_all_t & qos_print_all,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378)

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
```
to_string(qos, rti::core::qos_print_all);
```

Returns

The string representation of the qos

7.9.2.4 operator<<() [1/2]

```
std::ostream & operator<< (
            std::ostream & out,
            const rti::domain::qos::DomainParticipantFactoryQos & qos )  [inline]
```

<<***extension***>> *(p. 354)* Prints a dds::sub::qos::DomainParticipantFactoryQos to an output stream.

7.9.2.5 to_string() [4/6]

```
std::string to_string (
            const  DomainParticipantQos & qos,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::domain::qos::DomainParticipantQos** (p. 1384)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
DomainParticipantQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for DomainParticipantQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
DomainParticipantQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

7.9.2.6 to_string() [5/6]

```
std::string to_string (
            const  DomainParticipantQos & qos,
            const  DomainParticipantQos & base,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::domain::qos::DomainParticipantQos** (p. 1384)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

7.9.2.7 to_string() [6/6]

```
std::string to_string (
            const  DomainParticipantQos & qos,
            const  rti::core::qos_print_all_t & qos_print_all,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(*p. *354)* Obtains a string representation of the **dds::domain::qos::DomainParticipantQos** (p. 1384)

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
```
to_string(qos, rti::core::qos_print_all);
```

Returns

The string representation of the qos

7.9.2.8 operator<<() [2/2]

```
std::ostream & operator<< (
            std::ostream & out,
            const rti::domain::qos::DomainParticipantQos & qos )  [inline]
```

<<***extension***>> *(*p. *354)* Prints a dds::sub::qos::DomainParticipantQos to an output stream.

7.10 dds::pub Namespace Reference

Contains the type and functions to support publishing topics.

Namespaces

- namespace **qos**

  Contains ***PublisherQos*** *(*p. *2007) and* ***DataWriterQos*** *(*p. *1226).*

Classes

- class **AnyDataWriter**

  <<***reference-type***>> (p. *351) This class provides an non-template holder for representing a* ***DataWriter*** (p. *1138) of any type*
- class **AnyDataWriterListener**

  The listener to notify status changes for a ***dds::pub::DataWriter*** (p. *1138) of a generic type.*
- class **CoherentSet**

  <<***value-type***>> (p. *350) A publishing application can request that a set of DDS data-sample changes be propagated in such a way that they are interpreted at the receivers' side as a cohesive set of modifications.*
- class **DataWriter**

  <<***reference-type***>> (p. *351) Allows an application to publish data for a* ***dds::topic::Topic*** (p. *2474)*
- class **DataWriterListener**

  The ***Listener*** (p. *1637) to notify status changes for a* ***dds::pub::DataWriter*** (p. *1138).*
- class **NoOpDataWriterListener**

  A convenience implementation of ***DataWriterListener*** (p. *1202) where all methods are overridden to do nothing.*
- class **NoOpPublisherListener**

  A convenience implementation of ***PublisherListener*** (p. *2006) where all methods are overridden to do nothing.*
- class **Publisher**

  <<***reference-type***>> (p. *351) A publisher is the object responsible for the actual dissemination of publications.*
- class **PublisherListener**

  The listener to notify status changes for a ***dds::pub::Publisher*** (p. *1993).*
- class **SuspendedPublication**

  <<***value-type***>> (p. *350) Indicates that the application is about to make multiple modifications using several* ***dds::pub::DataWriter*** (p. *1138)'s belonging to the same* ***dds::pub::Publisher*** (p. *1993)*

Functions

- template<typename T >
  DataWriter< T > **get** (const **AnyDataWriter** &any_writer)

  Same as ***AnyDataWriter::get()*** (p. *819)*
- void **ignore** (**dds::domain::DomainParticipant** &participant, const **dds::core::InstanceHandle** &handle)

  Instructs RTI Connext to locally ignore a publication.
- template<typename FwdIterator >
  void **ignore** (**dds::domain::DomainParticipant** &participant, FwdIterator begin, FwdIterator end)

  Instructs RTI Connext to locally ignore several publications.
- template<typename T >
  dds::core::InstanceHandleSeq **matched_subscriptions** (const **dds::pub::DataWriter**< T > &writer)

  Retrieve the list of subscriptions currently associated with a ***dds::pub::DataWriter*** (p. *1138).*
- template<typename T , typename FwdIterator >
  FwdIterator **matched_subscriptions** (const **dds::pub::DataWriter**< T > &writer, FwdIterator begin, FwdIterator end)

  Retrieve the list of subscriptions currently associated with a ***dds::pub::DataWriter*** (p. *1138).*
- template<typename T >
  const **dds::topic::SubscriptionBuiltinTopicData** **matched_subscription_data** (const **dds::pub::DataWriter**< T > &writer, const **dds::core::InstanceHandle** &subscription_handle)

  Retrieves information on a subscription that is currently associated with a ***dds::pub::DataWriter*** (p. *1138).*

- template<typename WRITER , typename FwdIterator >
  uint32_t **find** (const **dds::pub::Publisher** &publisher, RTI_STRING_IN topic_name, FwdIterator begin, uint32_t max_size)

  *Retrieves the **dds::pub::DataWriter*** (p. *1138) for a specific topic name.*
- template<typename WRITER , typename BinIterator >
  uint32_t **find** (const **dds::pub::Publisher** &publisher, RTI_STRING_IN topic_name, BinIterator begin)

  *Retrieves the **dds::pub::DataWriter*** (p. *1138) for a specific topic name.*

7.10.1 Detailed Description

Contains the type and functions to support publishing topics.

7.10.2 Function Documentation

7.10.2.1 get()

```
template<typename T >
DataWriter< T > get (
             const  AnyDataWriter & any_writer )
```

Same as **AnyDataWriter::get()** (p. 819)

7.10.2.2 ignore() [1/2]

```
void ignore (
              dds::domain::DomainParticipant & participant,
             const  dds::core::InstanceHandle & handle )
```

Instructs RTI Connext to locally ignore a publication.

A publication is defined by the association of a topic name, user data, and partition set on the **dds::pub::Publisher** (p. 1993) (see **dds::topic::PublicationBuiltinTopicData** (p. 1977)). After this call, any data written by that publication's **dds::pub::DataWriter** (p. 1138) will be ignored.

This operation can be used to ignore local *and* remote DataWriters.

The publication (**DataWriter** (p. 1138)) to ignore is identified by the `handle` argument.

- To ignore a *remote* **DataWriter** (p. 1138), the `handle` can be obtained from the **dds::sub::SampleInfo** (p. 2285) retrieved when reading data samples from the built-in **dds::sub::DataReader** (p. 976) for the publication topic.
- To ignore a *local* **DataWriter** (p. 1138), the `handle` can be obtained by calling **dds::core::Entity::instance_handle()** (p. 1525) for the local **DataWriter** (p. 1138).

There is no way to reverse this operation.

Parameters

participant	The DomainParticipant where the publication will be ignored
handle	<<***in***>> *(p. 355)* Handle of the **dds::pub::DataWriter** (p. 1138) to be ignored.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::OutOfResourcesError** (p. 1898) or **dds::core::NotEnabledError** (p. 1869)

See also

dds::topic::PublicationBuiltinTopicData (p. 1977)

dds::topic::publication_topic_name() (p. 443)

dds::sub::builtin_subscriber (p. 671)

Referenced by **dds::pub::DataWriter< T >::ignore()**.

7.10.2.3 ignore() [2/2]

```
template<typename FwdIterator >
void ignore (
              dds::domain::DomainParticipant & participant,
             FwdIterator begin,
             FwdIterator end )
```

Instructs RTI Connext to locally ignore several publications.

Template Parameters

FwdIterator	A forward iterator whose value type is **dds::core::InstanceHandle** (p. 1612)

See also

ignore(dds::domain::DomainParticipant&, const dds::core::InstanceHandle&) (p. 647);

7.10.2.4 matched_subscriptions() [1/2]

```
template<typename T >
dds::core::InstanceHandleSeq matched_subscriptions (
             const  dds::pub::DataWriter< T > & writer )
```

Retrieve the list of subscriptions currently associated with a **dds::pub::DataWriter** (p. 1138).

A subscription is considered to be matching if all of the following criteria are true:

- The subscription is within the same domain as this publication.
- The subscription has a matching **dds::topic::Topic** (p. 2474).
- The subscription has compatible QoS.
- If the applications are using partitions, the subscription shares a common partition with this publication.
- The **dds::domain::DomainParticipant** (p. 1326) has not indicated that the subscription's **dds::domain::DomainParticipant** (p. 1326) should be "ignored" by means of the **dds::pub::ignore** (p. 647) API.
- If the publication is using the **rti::core::policy::MultiChannel** (p. 1749) and the subscription is using a **dds::topic::ContentFilteredTopic** (p. 954), there is an intersection between at least one of the associated filters.
- If the endpoints need to exchange key material to communicate (i.e., they are securing their communications), the writer has completed the key exchange with reader.

The handles returned list are the ones that RTI Connext uses to locally identify the corresponding matched **dds::sub::DataReader** (p. 976) entities. These handles match the ones that appear in the **dds::sub::SampleInfo::instance_handle** (p. 2290) field of the **dds::sub::SampleInfo** (p. 2285) when reading the **dds::topic::subscription_topic_name()** (p. 444) builtin topic.

This API may return the subscription handles of subscriptions that are inactive. **dds::pub::DataWriter::is_matched_subscription_active** (p. 1191) can be used to check this.

The maximum number of matches possible is configured with **rti::core::policy::DomainParticipantResourceLimits** (p. 1391). .

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::NotEnabledError** (p. 1869)

Referenced by **dds::pub::DataWriter< T >::matched_subscriptions()**.

7.10.2.5 matched_subscriptions() [2/2]

```
template<typename T , typename FwdIterator >
FwdIterator matched_subscriptions (
            const  dds::pub::DataWriter< T > & writer,
            FwdIterator begin,
            FwdIterator end )
```

Retrieve the list of subscriptions currently associated with a **dds::pub::DataWriter** (p. 1138).

This operation is similar to **matched_subscriptions(const dds::pub::DataWriter<T>&)** (p. 648) but it copies the instance handles into an iterator range.

Template Parameters

FwdIterator	A forward iterator whose value type is **dds::core::InstanceHandle** (p. 1612)

7.10.2.6 matched_subscription_data()

```
template<typename T >
const  dds::topic::SubscriptionBuiltinTopicData matched_subscription_data (
             const  dds::pub::DataWriter< T > & writer,
             const  dds::core::InstanceHandle & subscription_handle )
```

Retrieves information on a subscription that is currently associated with a **dds::pub::DataWriter** (p. 1138).

The `subscription_handle` must correspond to a subscription currently associated with the **dds::pub::DataWriter** (p. 1138). Otherwise, the operation will fail and fail with **dds::core::InvalidArgumentError** (p. 1619). Use **dds::pub::matched_subscriptions()** (p. 648) to find the subscriptions that are currently matched with the **dds::pub::DataWriter** (p. 1138).

The above information is also available through **dds::sub::DataReaderListener::on_data_available()** (p. 1058) (if a reader listener is installed on the dds::sub::DataReader<dds::topic::SubscriptionBuiltinTopicData>).

When the subscription data is updated, for example when the content filter property changes, there is a small window of time in between when the **DataWriter** (p. 1138) is made aware of these changes and when they actually take effect. Taking effect in this example means that the **DataWriter** (p. 1138) will perform writer-side filtering using the new filter property values (filter expression and/or parameters).

When the **DataWriter** (p. 1138) is made aware of the changes they will first be seen in the **dds::sub::DataReaderListener::on_data_available()** (p. 1058) of the dds::sub::DataReader<dds::topic::SubscriptionBuiltinTopicData>. When these changes are applied, they will be seen in the output of this API because this API blocks until the most recent changes known to the **DataWriter** (p. 1138) have taken effect. This API will only block when called outside of a listener callback, in order to not block the internal threads from making progress.

If application behavior depends on being made aware of information about a subscription only after it has taken effect on the **DataWriter** (p. 1138), the recommended pattern for usage of this API is to wait for subscription data to be received either through polling this API or by installing a listener on the dds::sub::DataReader<dds::topic::SubscriptionBuiltinTopicData>. When a new sample is received by the builtin DataReader, this API may be called in a separate thread and will return the expected matched subscription data once it has been applied to the **DataWriter** (p. 1138).

Because this API blocks, it is possible for this API to time out while waiting for the changes to be applied. A timeout may happen if the DataReader's subscription data is changing rapidly, preventing the **DataWriter** (p. 1138) from returning valid information before newer data has been received, or if an application is performing a task in a listener callback, thereby preventing the middleware's threads from executing events in a timely manner.

Note: This operation does not retrieve the **dds::topic::SubscriptionBuiltinTopicData::type** (p. 2434). This information is available through **dds::sub::DataReaderListener::on_data_available()** (p. 1058) (if a reader listener is installed on the dds::sub::DataReader<dds::topic::SubscriptionBuiltinTopicData>).

Parameters

writer	The **DataWriter** (p. 1138) to which the subscription is associated
subscription_handle	*<<**in**>> (p. 355).* Handle to a specific subscription associated with the **dds::sub::DataReader** (p. 976). Must correspond to a subscription currently associated with the **dds::pub::DataWriter** (p. 1138).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::NotEnabledError** (p. 1869), or **dds::core::TimeoutError** (p. 2473)

7.10.2.7 find() [1/2]

```
template<typename WRITER , typename FwdIterator >
uint32_t find (
             const  dds::pub::Publisher & publisher,
             RTI_STRING_IN topic_name,
             FwdIterator begin,
             uint32_t max_size )
```

Retrieves the **dds::pub::DataWriter** (p. 1138) for a specific topic name.

This returned **dds::pub::DataWriter** (p. 1138) is either enabled or disabled.

If more than one **dds::pub::DataWriter** (p. 1138) is attached to the **dds::pub::Publisher** (p. 1993) with the same `topic_name`, then this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **dds::pub::DataWriter** (p. 1138) in one thread while another thread is simultaneously creating or destroying that **dds::pub::DataWriter** (p. 1138).

This function retrieves a previously-created **DataWriter** (p. 1138) belonging to the **dds::pub::Publisher** (p. 1993) that is attached to a **dds::topic::Topic** (p. 2474) with a matching topic name. If no such **DataWriter** (p. 1138) exists, the operation will return an empty container. The use of this operation on the built-in **Publisher** (p. 1993) allows access to the built-in **DataWriter** (p. 1138) entities for the built-in topics

Template Parameters

Writer	The type of the writer. It can be **dds::pub::AnyDataWriter** (p. 815), or an instantiation of dds::pub::DataWriter<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))
FwdIterator	The type of forward iterator passed to this function

Parameters

publisher	The **dds::pub::Publisher** (p. 1993) the **DataWriter** (p. 1138) belongs to
topic_name	Name of the **dds::topic::Topic** (p. 2474) associated with the **DataWriter** (p. 1138) that is to be looked up.
begin	A forward iterator to the position in the destination container to insert the found **DataWriter** (p. 1138) in
max_size	Only 1 **DataWriter** (p. 1138) will be returned from this function

Returns

The number of DataWriters that were found (either 0 or 1)

7.10.2.8 find() [2/2]

```
template<typename WRITER , typename BinIterator >
uint32_t find (
             const  dds::pub::Publisher & publisher,
             RTI_STRING_IN topic_name,
             BinIterator begin )
```

Retrieves the **dds::pub::DataWriter** (p. 1138) for a specific topic name.

This returned **dds::pub::DataWriter** (p. 1138) is either enabled or disabled.

If more than one **dds::pub::DataWriter** (p. 1138) is attached to the **dds::pub::Publisher** (p. 1993) with the same `topic_name`, then this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **dds::pub::DataWriter** (p. 1138) in one thread while another thread is simultaneously creating or destroying that **dds::pub::DataWriter** (p. 1138).

This function retrieves a previously created **DataWriter** (p. 1138) belonging to the **dds::pub::Publisher** (p. 1993) that is attached to a **dds::topic::Topic** (p. 2474) with a matching topic name. If no such **DataWriter** (p. 1138) exists, the operation will return an empty container. The use of this operation on the built-in **Publisher** (p. 1993) allows access to the built-in **DataWriter** (p. 1138) entities for the built-in topics

Template Parameters

Writer	The type of the writer. It can be **dds::pub::AnyDataWriter** (p. 815), or an instantiation of dds::pub::DataWriter<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))
BinIterator	BinIterator The type of back-inserting iterator passed to this function

Parameters

publisher	The **dds::pub::Publisher** (p. 1993) the **DataWriter** (p. 1138) belongs to
topic_name	Name of the **dds::topic::Topic** (p. 2474) associated with the **DataWriter** (p. 1138) that is to be looked up.
begin	A back-inserting iterator to the position in the destination container to insert the found **DataWriter** (p. 1138) into

Returns

The number of DataWriters that were found (either 0 or 1)

7.11 dds::pub::qos Namespace Reference

Contains **PublisherQos** (p. 2007) and **DataWriterQos** (p. 1226).

Classes

- class **DataWriterQos**

  <<***value-type***>> *(p. 350) Container of the QoS policies that a* ***dds::pub::DataWriter*** *(p. 1138) supports*
- class **PublisherQos**

  <<***value-type***>> *(p. 350) Container of the QoS policies that a* ***dds::pub::Publisher*** *(p. 1993) supports*

Functions

- std::string **to_string** (const **DataWriterQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::pub::qos::DataWriterQos*** *(p. 1226)*
- std::string **to_string** (const **DataWriterQos** &qos, const **DataWriterQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::pub::qos::DataWriterQos*** *(p. 1226)*
- std::string **to_string** (const **DataWriterQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the dds::sub::qos::DataWriterQos*
- std::ostream & **operator**<< (std::ostream &out, const rti::pub::qos::DataWriterQos &qos)

  <<***extension***>> *(p. 354) Prints a* ***dds::pub::qos::DataWriterQos*** *(p. 1226) to an output stream.*
- void **swap** (PublisherQosImpl &left, PublisherQosImpl &right) OMG_NOEXCEPT

  Swap the contents of two ***PublisherQos*** *(p. 2007) objects.*
- std::string **to_string** (const **PublisherQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::pub::qos::PublisherQos*** *(p. 2007)*
- std::string **to_string** (const **PublisherQos** &qos, const **PublisherQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::pub::qos::PublisherQos*** *(p. 2007)*
- std::string **to_string** (const **PublisherQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::pub::qos::PublisherQos*** *(p. 2007)*
- std::ostream & **operator**<< (std::ostream &out, const rti::pub::qos::PublisherQos &qos)

  <<***extension***>> *(p. 354) Prints a* ***dds::pub::qos::PublisherQos*** *(p. 2007) to an output stream.*

7.11.1 Detailed Description

Contains **PublisherQos** (p. 2007) and **DataWriterQos** (p. 1226).

7.11.2 Function Documentation

7.11.2.1 to_string() [1/6]

```
std::string to_string (
            const  DataWriterQos & qos,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::pub::qos::DataWriterQos** (p. 1226)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
DataWriterQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for DataWriterQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
DataWriterQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

7.11.2.2 to_string() [2/6]

```
std::string to_string (
             const  DataWriterQos & qos,
             const  DataWriterQos & base,
             const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::pub::qos::DataWriterQos** (p. 1226)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

7.11.2.3 to_string() [3/6]

```
std::string to_string (
             const  DataWriterQos & qos,
             const  rti::core::qos_print_all_t & qos_print_all,
             const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the dds::sub::qos::DataWriterQos

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
`to_string(qos, rti::core::qos_print_all);`

Returns

The string representation of the qos

7.11.2.4 operator<<() [1/2]

```
std::ostream & operator<< (
            std::ostream & out,
            const rti::pub::qos::DataWriterQos & qos )  [inline]
```

<<***extension***>> *(p. 354)* Prints a **dds::pub::qos::DataWriterQos** (p. 1226) to an output stream.

7.11.2.5 swap()

```
void swap (
            PublisherQosImpl & left,
            PublisherQosImpl & right )
```

Swap the contents of two **PublisherQos** (p. 2007) objects.

Parameters

left	A **PublisherQos** (p. 2007)
right	The other **PublisherQos** (p. 2007)

7.11.2.6 to_string() [4/6]

```
std::string to_string (
            const  PublisherQos & qos,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::pub::qos::PublisherQos** (p. 2007)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos

object.

```
PublisherQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for PublisherQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
PublisherQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

7.11.2.7 to_string() [5/6]

```
std::string to_string (
            const  PublisherQos & qos,
            const  PublisherQos & base,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::pub::qos::PublisherQos** (p. 2007)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

7.11.2.8 to_string() [6/6]

```
std::string to_string (
            const  PublisherQos & qos,
            const  rti::core::qos_print_all_t & qos_print_all,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::pub::qos::PublisherQos** (p. 2007)

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
`to_string(qos, rti::core::qos_print_all);`

Returns

The string representation of the qos

7.11.2.9 operator<<() [2/2]

```
std::ostream & operator<< (
            std::ostream & out,
            const rti::pub::qos::PublisherQos & qos )  [inline]
```

<<***extension***>> *(p. 354)* Prints a **dds::pub::qos::PublisherQos** (p. 2007) to an output stream.

7.12 dds::sub Namespace Reference

Contains the types and functions to support subscribing to topics.

Namespaces

- namespace **cond**

  Contains Conditions specific to DataReaders.
- namespace **qos**

  *Contains **DataReaderQos** (p. 1072) and **SubscriberQos** (p. 2423).*
- namespace **status**

  *Contains **DataState** (p. 1118).*

Classes

- class **AnyDataReader**

  *<<**reference-type**>> (p. 351) This class provides an non-template holder for representing a **DataReader** (p. 976) of any type*
- class **AnyDataReaderListener**

  *The listener to notify status changes for a **dds::sub::DataReader** (p. 976) of a generic type.*
- class **CoherentAccess**

  *<<**value-type**>> (p. 350) Controls whether RTI Connext will preserve the groupings of changes made by the publishing application by means of begin_coherent_changes and end_coherent_changes.*
- class **DataReader**

  *<<**reference-type**>> (p. 351) Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **dds::sub::Subscriber** (p. 2410).*
- class **DataReaderListener**

  *The **Listener** (p. 1637) to notify status changes for a **dds::sub::DataReader** (p. 976).*
- class **GenerationCount**

  *<<**value-type**>> (p. 350)*
- class **LoanedSamples**

  *<<**move-only-type**>> (p. 353) Provides temporary access to a collection of samples (data and info) from a **DataReader** (p. 976).*
- class **NoOpDataReaderListener**

  *A convenience implementation of **DataReaderListener** (p. 1055) where all methods are overridden to do nothing.*
- class **NoOpSubscriberListener**

  *A convenience implementation of **SubscriberListener** (p. 2422) where all methods are overridden to do nothing.*
- class **Query**

  *<<**value-type**>> (p. 350) Encapsulates a query for a **dds::sub::cond::QueryCondition** (p. 2059).*
- class **Rank**

  *<<**value-type**>> (p. 350) Contains the sample and generation ranks of a data-sample*
- class **ReadModeDummyType**
- class **Sample**

  *<<**value-type**>> (p. 350) This class encapsulate the data and meta-data associated with DDS samples.*
- class **SampleInfo**

  *<<**value-type**>> (p. 350) Information that accompanies each sample received by a **DataReader** (p. 976)*
- class **SharedSamples**

  *<<**reference-type**>> (p. 351) A sharable and container-safe version of **LoanedSamples** (p. 1663).*
- class **Subscriber**

  *<<**reference-type**>> (p. 351) A subscriber is the object responsible for actually receiving data from a subscription.*
- class **SubscriberListener**

  *The listener to notify status changes for a **dds::sub::Subscriber** (p. 2410).*

Functions

- template<typename T >
  DataReader< T > **get** (const **AnyDataReader** &any_reader)

  *Same as **AnyDataReader::get()** (p. 810)*
- bool **read** (**dds::sub::ReadModeDummyType**)

  The stream manipulator to indicate that the reader should read samples as opposed to taking the samples.
- bool **take** (**dds::sub::ReadModeDummyType**)

  The stream manipulator to indicate that the reader should take samples as opposed to reading the samples.
- dds::sub::functors::MaxSamplesManipulatorFunctor **max_samples** (uint32_t n)

  Stream manipulator to set the maximum number of samples to read or take.
- dds::sub::functors::ContentFilterManipulatorFunctor **content** (const **dds::sub::Query** &query)

  *Stream manipulator to set a **Query** (p. 2053) to use during the subsequent read/take operation.*
- dds::sub::functors::ConditionManipulatorFunctor **condition** (const **dds::sub::cond::ReadCondition** &condition)

  Stream manipulator to set a QueryCondition to use during the subsequent read/take operation.
- dds::sub::functors::StateFilterManipulatorFunctor **state** (const **dds::sub::status::DataState** &s)

  Stream manipulator to specify the DataState of the samples that should be read/taken.
- dds::sub::functors::InstanceManipulatorFunctor **instance** (const **dds::core::InstanceHandle** &h)

  Stream manipulator to specify the instance whose samples should be read or taken.
- dds::sub::functors::NextInstanceManipulatorFunctor **next_instance** (const **dds::core::InstanceHandle** &h)

  Stream manipulator to specify the samples belonging to the 'next' instance after the provided instance handle should be accessed.
- void **ignore** (**dds::domain::DomainParticipant** &participant, const **dds::core::InstanceHandle** &handle)

  Instructs RTI Connext to locally ignore a subscription.
- template<typename FwdIterator >
  void **ignore** (**dds::domain::DomainParticipant** &participant, FwdIterator **begin**, FwdIterator **end**)

  Instructs RTI Connext to locally ignore subscriptions.
- template<typename T >
  const **::dds::core::InstanceHandleSeq** **matched_publications** (const **dds::sub::DataReader**< T > &reader)

  *Retrieve the list of publications currently "associated" with a **DataReader** (p. 976).*
- template<typename T , typename FwdIterator >
  FwdIterator **matched_publications** (const **dds::sub::DataReader**< T > &reader, FwdIterator **begin**, FwdIterator **end**)

  *Retrieve the list of publications currently "associated" with a **DataReader** (p. 976).*
- template<typename T >
  const **dds::topic::PublicationBuiltinTopicData** **matched_publication_data** (const **dds::sub::DataReader**< T > &reader, const **dds::core::InstanceHandle** &handle)

  *This operation retrieves the information on a publication that is currently "associated" with the **DataReader** (p. 976).*
- **dds::sub::Subscriber** **builtin_subscriber** (const **dds::domain::DomainParticipant** &dp)

  *Access the built-in **Subscriber** (p. 2410).*
- template<typename READER , typename FwdIterator >
  uint32_t **find** (const **dds::sub::Subscriber** &subscriber, RTI_STRING_IN topic_name, FwdIterator **begin**, uint32_t max_size)

  *This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching topic name.*
- template<typename READER , typename BinIterator >
  uint32_t **find** (const **dds::sub::Subscriber** &subscriber, RTI_STRING_IN topic_name, BinIterator **begin**)

  *This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching topic name.*

- template<typename READER , typename T , typename FwdIterator >
  uint32_t **find** (const **dds::sub::Subscriber** &sub, const **dds::topic::TopicDescription**< T > &topic_description, FwdIterator **begin**, uint32_t max_size)
- template<typename READER , typename T , typename BinIterator >
  uint32_t **find** (const **dds::sub::Subscriber** &subscriber, const **dds::topic::TopicDescription**< T > &topic_description, BinIterator **begin**)

  *This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching TopicDescription.*
- template<typename AnyDataReaderFwdIterator >
  uint32_t **find** (const **dds::sub::Subscriber** &subscriber, const **dds::sub::status::DataState** &data_state, AnyDataReaderFwdIterator **begin**, uint32_t max_size)

  *Allows the application to access the AnyDataReaders that contain samples with the specified **dds::sub::status::DataState** (p. 1118).*
- template<typename AnyDataReaderBackInsertIterator >
  uint32_t **find** (const **dds::sub::Subscriber** &subscriber, const **dds::sub::status::DataState** &data_state, AnyDataReaderBackInsertIterator **begin**)

  *Allows the application to access the AnyDataReaders that contain samples with the specified **dds::sub::status::DataState** (p. 1118).*
- template<typename T >
  LoanedSamples< T > **move** (**LoanedSamples**< T > &ls) OMG_NOEXCEPT

  *Creates a new **LoanedSamples** (p. 1663) instance by moving the contents of an existing one.*
- template<typename T >
  LoanedSamples< T >::iterator **begin** (**LoanedSamples**< T > &ls)
- template<typename T >
  LoanedSamples< T >::const_iterator **begin** (const **LoanedSamples**< T > &ls)
- template<typename T >
  LoanedSamples< T >::iterator **end** (**LoanedSamples**< T > &ls)
- template<typename T >
  LoanedSamples< T >::const_iterator **end** (const **LoanedSamples**< T > &ls)
- template<typename T >
  void **swap** (**LoanedSamples**< T > &ls1, **LoanedSamples**< T > &ls2) throw ()

7.12.1 Detailed Description

Contains the types and functions to support subscribing to topics.

7.12.2 Function Documentation

7.12.2.1 get()

```
template<typename T >
DataReader< T > get (
            const  AnyDataReader & any_reader )
```

Same as **AnyDataReader::get()** (p. 810)

7.12.2.2 read()

```
bool read (
            dds::sub::ReadModeDummyType  )
```

The stream manipulator to indicate that the reader should read samples as opposed to taking the samples.

Usage:
```
reader » read » loaned_samples;
```

See also

dds::sub::DataReader::operator >>(bool(*manipulator)(**ReadModeDummyType** (p. 2144)))

7.12.2.3 take()

```
bool take (
            dds::sub::ReadModeDummyType  )
```

The stream manipulator to indicate that the reader should take samples as opposed to reading the samples.

Usage:
```
reader » take » loaned_samples;
```

The default mode to access samples is to take, so the above is equivalent to:
```
reader » loaned_samples;
```

See also

dds::sub::DataReader::operator >>(bool(*manipulator)(**ReadModeDummyType** (p. 2144)))

7.12.2.4 max_samples()

```
dds::sub::functors::MaxSamplesManipulatorFunctor max_samples (
            uint32_t n )  [inline]
```

Stream manipulator to set the maximum number of samples to read or take.

Use this function to set the maximum number of samples to read/take by passing it to the **DataReader::operator >>(Functor f)** (p. 995) operator.

Parameters

n	The maximum number of samples to take

See also

dds::sub::DataReader::operator >>(Functor f) (p. 995)

7.12.2.5 content()

```
dds::sub::functors::ContentFilterManipulatorFunctor content (
            const  dds::sub::Query & query )  [inline]
```

Stream manipulator to set a **Query** (p. 2053) to use during the subsequent read/take operation.

The effect of using this manipulator is that the subsequent read/take will filter the samples based on the **Query** (p. 2053)'s expression. If the **DataReader** (p. 976) has no samples that meet the constraints, the read/take will not return any data.

If this stream manipulator comes before a call to the condition(const dds::sub::cond::QueryCondition& query_condition) manipulator then it will be overridden and will not have any effect on the read or take operation. Similarly, if this operation follows a call to **condition()** (p. 663), then the previously set QueryCondition will be cleared.

Parameters

query	The **Query** (p. 2053) to use during the read/take

See also

dds::sub::DataReader::operator >>(Functor f) (p. 995)

dds::sub::condition(const dds::sub::cond::QueryCondition& query_condition)

7.12.2.6 condition()

```
dds::sub::functors::ConditionManipulatorFunctor condition (
            const  dds::sub::cond::ReadCondition & condition )  [inline]
```

Stream manipulator to set a QueryCondition to use during the subsequent read/take operation.

The effect of using this manipulator is that the subsequent read/take will filter the samples based on the Query↩ Conditions's expression and state. If the **DataReader** (p. 976) has no samples that meet the constraints, the read/take will not return any data.

If this stream manipulator comes before a call to the **content(const dds::sub::Query& query)** (p. 663) manipulator then it will be overridden and will not have any effect on the read or take operation. Similarly, if this operation follows a call to **content()** (p. 663) and/or **state(const dds::sub::status::DataState& s)** (p. 664), then the previously set **Query** (p. 2053) and DataState will be cleared.

This manipulator is effectively a combination of the content and state manipulators.

For example:

```
reader » read
       » content(dds::sub::Query(system.reader, "foo = 7"))
       » state(dds::sub::status::DataState::new_data())
       » samples;
```

is equivalent to:

```
reader » read
       » condition(Query(system.reader, "foo = 7"), DataState()::new_data())
       » samples;
```

Parameters

condition	The QueryCondition to use during the read/take

See also

dds::sub::DataReader::operator >>(Functor f) (p. 995)

content(const dds::sub::Query& query) (p. 663)

Referenced by **dds::sub::DataReader< T >::condition()**, and **dds::sub::cond::ReadCondition::create_read_↩ condition_ex()**.

7.12.2.7 state()

```
dds::sub::functors::StateFilterManipulatorFunctor state (
             const  dds::sub::status::DataState & s )  [inline]
```

Stream manipulator to specify the DataState of the samples that should be read/taken.

By setting the **dds::sub::status::DataState** (p. 1118) you can specify the state of the samples that should be read or taken. The DataState of a sample encapsulates the **dds::sub::status::SampleState** (p. 2316), **dds::sub::status::↩ ViewState** (p. 2614), and **dds::sub::status::InstanceState** (p. 1615) of a sample.

If this stream manipulator comes before a call to the condition(const dds::sub::cond::QueryCondition& query_condition) manipulator then it will be overridden and will not have any effect on the read or take operation.

Parameters

s	The DataState of the samples to be read or taken

See also

dds::sub::DataReader::operator >>(Functor f) (p. 995)

7.12.2.8 instance()

```
dds::sub::functors::InstanceManipulatorFunctor instance (
            const  dds::core::InstanceHandle & h )  [inline]
```

Stream manipulator to specify the instance whose samples should be read or taken.

This operation causes the subsequent read or take operation to access only samples belonging the single specified instance whose handle is `h`.

Upon successful completion, the data collection will contain samples all belonging to the same instance. The corresponding **SampleInfo** (p. 2285) verifies **SampleInfo.instance_handle()** (p. 2290) == `h`.

The subsequent read/take operation will be semantically equivalent to a read or take without specifying the instance, except in building the collection, the **DataReader** (p. 976) will check that the sample belongs to the specified instance and otherwise it will not place the sample in the returned collection.

The subsequent read/take may operation may fail with **dds::core::InvalidArgumentError** (p. 1619) if the Instance↩Handle does not correspond to an existing data-object known to the **DataReader** (p. 976).

Parameters

h	The handle of the instance to access

See also

dds::sub::DataReader::operator >>(Functor f) (p. 995)

7.12.2.9 next_instance()

```
dds::sub::functors::NextInstanceManipulatorFunctor next_instance (
            const  dds::core::InstanceHandle & h )  [inline]
```

Stream manipulator to specify the samples belonging to the 'next' instance after the provided instance handle should be accessed.

This operation causes the subsequent read or take operation to access only samples belonging a single instance whose handle is considered 'next' after the provided InstanceHandle `h`.

The accessed samples will all belong to the 'next' instance with InstanceHandle 'greater' than the specified previous handle that has available samples.

This operation implies the existence of a total order 'greater-than' relationship between the instance handles. The specifics of this relationship are not all important and are implementation specific. The important thing is that, according to the middleware, all instances are ordered relative to each other. This ordering is between the instance handles; It should not depend on the state of the instance (e.g. whether it has data or not) and must be defined even for instance handles that do not correspond to instances currently managed by the **dds::sub::DataReader** (p. 976). For the purposes of the ordering, it should be 'as if' each instance handle was represented as unique integer.

The behavior of **dds::sub::DataReader::Selector::next_instance** (p. 2322) is 'as if' the **dds::sub::DataReader** (p. 976) invoked **dds::sub::instance(const dds::core::InstanceHandle& h)** (p. 665), passing the smallest instance_handle among all the ones that: (a) are greater than `previous_handle`, and (b) have available samples (i.e. samples that meet the constraints imposed by the specified states).

The special value **dds::core::InstanceHandle::nil()** (p. 1614) is guaranteed to be 'less than' any valid `instance_handle`. So the use of the parameter value `previous_handle` == **dds::core::InstanceHandle::nil()** (p. 1614) will return the samples for the instance which has the smallest `instance_handle` among all the instances that contain available samples.

Note

The operation **dds::sub::DataReader::Selector::next_instance** (p. 2322) is intended to be used in an application-driven iteration, where the application starts by passing `previous_handle` == **dds::core::InstanceHandle::nil()** (p. 1614), examines the samples returned, and then uses the `instance_handle` returned in the **dds::sub::SampleInfo** (p. 2285) as the value of the `previous_handle` argument to the next call to **dds::sub::DataReader::Selector::next_instance** (p. 2322). The iteration continues until the read/take operation doesn't return any more samples. This application-driven iteration is required to ensure that all samples on the reader queue are read.

Note that it is possible to call the **dds::sub::DataReader::Selector::next_instance** (p. 2322) operation with a `previous_handle` that does not correspond to an instance currently managed by the **dds::sub::DataReader** (p. 976). This is because as stated earlier the 'greater-than' relationship is defined even for handles not managed by the **dds::sub::DataReader** (p. 976). One practical situation where this may occur is when an application is iterating though all the instances, takes all the samples of a **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) instance, returns the loan (at which point the instance information may be removed, and thus the handle becomes invalid), and tries to read the next instance.

Parameters

h	The reference instance. The instance after this one will be selected

See also

dds::sub::DataReader::operator >>(Functor f) (p. 995)

7.12.2.10 ignore() [1/2]

```
void ignore (
              dds::domain::DomainParticipant & participant,
             const  dds::core::InstanceHandle & handle )
```

Instructs RTI Connext to locally ignore a subscription.

A subscription is defined by the association of a topic name, user data, and partition set on the **dds::sub::Subscriber** (p. 2410) (see **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)). After this call, any data received related to that subscription's **dds::sub::DataReader** (p. 976) will be ignored.

This operation can be used to ignore local *and* remote DataReaders.

The subscription to ignore is identified by the `handle` argument.

- To ignore a *remote* **DataReader** (p. 976), the `handle` can be obtained from the **dds::sub::SampleInfo** (p. 2285) retrieved when reading data samples from the built-in **dds::sub::DataReader** (p. 976) for the subscription topic.

- To ignore a *local* **DataReader** (p. 976), the `handle` can be obtained by calling **dds::core::Entity::instance_handle()** (p. 1525) for the local **DataReader** (p. 976).

There is no way to reverse this operation.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::OutOfResourcesError** (p. 1898) or **dds::core::NotEnabledError** (p. 1869)

See also

dds::topic::SubscriptionBuiltinTopicData (p. 2428)

dds::topic::subscription_topic_name() (p. 444)

dds::sub::builtin_subscriber (p. 671)

Parameters

participant	The DomainParticipant for which the remote entity will be ignored
handle	The InstanceHandle of the remote entity that has to be ignored

Referenced by **dds::sub::DataReader< T >::ignore()**.

7.12.2.11 ignore() [2/2]

```
template<typename FwdIterator >
void ignore (
              dds::domain::DomainParticipant & participant,
             FwdIterator begin,
             FwdIterator end )
```

Instructs RTI Connext to locally ignore subscriptions.

Parameters

participant	The DomainParticipant for which the remote entity will be ignored
begin	A forward iterator to the initial position in a **dds::core::InstanceHandleSeq** (p. 619) holding the handles to the remote DataReaders to be ignored.
end	A forward iterator to the final position in a **dds::core::InstanceHandleSeq** (p. 619) holding the handles to the remote DataReaders to be ignored.

See also

ignore(dds::domain::DomainParticipant&, const dds::core::InstanceHandle&) (p. 667);

7.12.2.12 matched_publications() [1/2]

```
template<typename T >
const  ::dds::core::InstanceHandleSeq matched_publications (
             const  dds::sub::DataReader< T > & reader )
```

Retrieve the list of publications currently "associated" with a **DataReader** (p. 976).

A publication is considered to be matching if all of the following criteria are true:

- The publication is within the same domain as this subscription.
- The publication has a matching **dds::topic::Topic** (p. 2474).
- The publication has compatible QoS.
- If the applications are using partitions, the publication shares a common partition with this subscription.
- The **dds::domain::DomainParticipant** (p. 1326) has not indicated that the publication's **dds::domain::DomainParticipant** (p. 1326) should be "ignored" by means of the **dds::pub::ignore** (p. 647) API.

- If the subscription is using a **dds::topic::ContentFilteredTopic** (p. 954) and the publication is using the **rti::core::policy::MultiChannel** (p. 1749), there is an intersection between at least one of the associated filters.
- If the endpoints need to exchange key material to communicate (i.e., they are securing their communications), the reader has completed the key exchange with the writer.

The handles returned in the `publication_handles` list are the ones that are used by the RTI Connext implementation to locally identify the corresponding matched **dds::pub::DataWriter** (p. 1138) entities. These handles match the ones that appear in the `instance_handle` field of the **dds::sub::SampleInfo** (p. 2285) when reading the **dds::topic::publication_topic_name()** (p. 443) builtin topic.

This API may return the publication handles of publications that are not alive. **dds::sub::DataReader::is_matched_publication_alive** (p. 1035) can be used to check the liveliness of the remote publication.

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::OutOfResourcesError** (p. 1898) if the sequence is too small and the system cannot resize it, or **dds::core::NotEnabledError** (p. 1869)

Template Parameters

T	The topic-type that the **DataReader** (p. 976) subscribes to

Parameters

reader	The reader whose publications are being retrieved

Returns

An InstanceHandleSeq containing the InstanceHandles of the matched publications for the provided **DataReader** (p. 976)

Referenced by **dds::sub::DataReader< T >::matched_publications()**.

7.12.2.13 matched_publications() [2/2]

```
template<typename T , typename FwdIterator >
FwdIterator matched_publications (
             const  dds::sub::DataReader< T > & reader,
             FwdIterator begin,
             FwdIterator end )
```

Retrieve the list of publications currently "associated" with a **DataReader** (p. 976).

A publication is considered to be matching if all of the following criteria are true:

- The publication is within the same domain as this subscription.
- The publication has a matching **dds::topic::Topic** (p. 2474).
- The publication has compatible QoS.
- If the applications are using partitions, the publication shares a common partition with this subscription.
- The **dds::domain::DomainParticipant** (p. 1326) has not indicated that the publication's **dds::domain::DomainParticipant** (p. 1326) should be "ignored" by means of the **dds::pub::ignore** (p. 647) API.
- If the subscription is using a **dds::topic::ContentFilteredTopic** (p. 954) and the publication is using the **rti::core::policy::MultiChannel** (p. 1749), there is an intersection between at least one of the associated filters.
- If the endpoints need to exchange key material to communicate (i.e., they are securing their communications), the reader has completed the key exchange with the writer.

The handles returned in the `publication_handles` list are the ones that are used by the RTI Connext implementation to locally identify the corresponding matched **dds::pub::DataWriter** (p. 1138) entities. These handles match the ones that appear in the `instance_handle` field of the **dds::sub::SampleInfo** (p. 2285) when reading the **dds::topic::publication_topic_name()** (p. 443) builtin topic.

This API may return the publication handles of publications that are not alive. **dds::sub::DataReader::is_matched_publication_alive** (p. 1035) can be used to check the liveliness of the remote publication.

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::OutOfResourcesError** (p. 1898) if the sequence is too small and the system cannot resize it, or **dds::core::NotEnabledError** (p. 1869)

Template Parameters

T	The topic-type that the **DataReader** (p. 976) subscribes to
FwdIterator	A forward iterator whose value type is **dds::core::InstanceHandle** (p. 1612)

Parameters

reader	The reader whose publications are being retrieved
begin	A forward iterator to the beginning position in the destination container of matching InstanceHandles
end	A forward iterator to the ending position in the destination container of matching InstanceHandles

Returns

An iterator placed at the last position in the container where a InstanceHandle was inserted

7.12.2.14 matched_publication_data()

```
template<typename T >
const  dds::topic::PublicationBuiltinTopicData matched_publication_data (
```

```
const  dds::sub::DataReader< T > & reader,
const  dds::core::InstanceHandle & handle )
```

This operation retrieves the information on a publication that is currently "associated" with the **DataReader** (p. 976).

The publication_handle must correspond to a publication currently associated with the **dds::sub::DataReader** (p. 976). Otherwise, the operation will fail with **dds::core::InvalidArgumentError** (p. 1619). Use the operation **dds::sub::matched_publications** (p. 668) to find the publications that are currently matched with the **dds::sub::DataReader** (p. 976).

Note: This operation does not retrieve the **dds::topic::PublicationBuiltinTopicData::type** (p. 1985). This information is available through **dds::sub::DataReaderListener::on_data_available()** (p. 1058) (if a reader listener is installed on the dds::sub::DataReader<dds::topic::PublicationBuiltinTopicData>).

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869)

Template Parameters

T	The topic-type that the **DataReader** (p. 976) subscribes to

Parameters

reader	The reader associated with the publication whose data is being retrieved
handle	The InstanceHandle Handle to a specific publication associated with the DataWriter. Must correspond to a publication currently associated with the **DataReader** (p. 976).

Returns

The **dds::topic::PublicationBuiltinTopicData** (p. 1977) of the publication that is associated with the provided handle

7.12.2.15 builtin_subscriber()

```
dds::sub::Subscriber builtin_subscriber (
            const  dds::domain::DomainParticipant & dp )
```

Access the built-in **Subscriber** (p. 2410).

Each **dds::domain::DomainParticipant** (p. 1326) contains several built-in **dds::topic::Topic** (p. 2474) objects as well as corresponding **dds::sub::DataReader** (p. 976) objects to access them. All of these **dds::sub::DataReader** (p. 976) objects belong to a single built-in **dds::sub::Subscriber** (p. 2410).

The built-in Topics are used to communicate information about other **dds::domain::DomainParticipant** (p. 1326), **dds::topic::Topic** (p. 2474), **dds::sub::DataReader** (p. 976), and **dds::pub::DataWriter** (p. 1138) objects.

The built-in subscriber is created when this operation is called for the first time. The built-in subscriber is deleted automatically when the **dds::domain::DomainParticipant** (p. 1326) is deleted.

Returns

The built-in **dds::sub::Subscriber** (p. 2410) singleton.

See also

dds::topic::SubscriptionBuiltinTopicData (p. 2428)

dds::topic::PublicationBuiltinTopicData (p. 1977)

dds::topic::ParticipantBuiltinTopicData (p. 1908)

dds::topic::TopicBuiltinTopicData (p. 2493)

Parameters

dp	The DomainParticipant that the built-in subscriber belongs to.

7.12.2.16 find() [1/6]

```
template<typename READER , typename FwdIterator >
uint32_t find (
             const  dds::sub::Subscriber & subscriber,
             RTI_STRING_IN topic_name,
             FwdIterator begin,
             uint32_t max_size )
```

This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching topic name.

Use this operation on the built-in **dds::sub::Subscriber** (p. 2410) (**Built-in Topics** (p. 46)) to access the built-in **dds::sub::DataReader** (p. 976) entities for the built-in topics.

The built-in **dds::sub::DataReader** (p. 976) is created when this operation is called on a built-in topic for the first time. The built-in **dds::sub::DataReader** (p. 976) is deleted automatically when the **dds::domain::DomainParticipant** (p. 1326) is deleted.

To ensure that builtin **dds::sub::DataReader** (p. 976) entities receive all the discovery traffic, it is suggested that you lookup the builtin **dds::sub::DataReader** (p. 976) before the **dds::domain::DomainParticipant** (p. 1326) is enabled. Looking up builtin **dds::sub::DataReader** (p. 976) may implicitly register builtin transports due to creation of **dds::sub::DataReader** (p. 976) (see **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered). Therefore, if you are want to modify builtin transport properties, do so *before* using this operation.

Therefore the suggested sequence when looking up builtin DataReaders is:

- Create a disabled **dds::domain::DomainParticipant** (p. 1326).
- (optional) Modify builtin transport properties
- Call **dds::sub::builtin_subscriber()** (p. 671).

- Call **dds::sub::find()** (p. 672).
- Call **enable()** (p. 581) on the DomainParticipant.

The returned **dds::sub::DataReader** (p. 976) may be enabled or disabled.

If more than one **dds::sub::DataReader** (p. 976) is attached to the **dds::sub::Subscriber** (p. 2410), this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **dds::sub::DataReader** (p. 976) in one thread while another thread is simultaneously creating or destroying that **dds::sub::DataReader** (p. 976).

Template Parameters

Reader	The type of the reader. It can be **dds::sub::AnyDataReader** (p. 807), or an instantiation of dds::sub::DataReader<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))
FwdIterator	The type of forward iterator passed to this function

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) the **DataReader** (p. 976) belongs to
topic_name	Name of the **dds::topic::Topic** (p. 2474) associated with the **DataReader** (p. 976) that is to be looked up.
begin	A forward iterator to the position in the destination container where the DataReaders will be copied into
max_size	Only 1 **DataReader** (p. 976) will be returned from this function

Returns

The number of DataReaders that were found (either 0 or 1)

Referenced by **dds::sub::DataReader< T >::find()**.

7.12.2.17 find() [2/6]

```
template<typename READER , typename BinIterator >
uint32_t find (
             const  dds::sub::Subscriber & subscriber,
             RTI_STRING_IN topic_name,
             BinIterator begin )
```

This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching topic name.

Use this operation on the built-in **dds::sub::Subscriber** (p. 2410) (**Built-in Topics** (p. 46)) to access the built-in **dds::sub::DataReader** (p. 976) entities for the built-in topics.

The built-in **dds::sub::DataReader** (p. 976) is created when this operation is called on a built-in topic for the first time. The built-in **dds::sub::DataReader** (p. 976) is deleted automatically when the **dds::domain::DomainParticipant** (p. 1326) is deleted.

To ensure that builtin **dds::sub::DataReader** (p. 976) entities receive all the discovery traffic, it is suggested that you lookup the builtin **dds::sub::DataReader** (p. 976) before the **dds::domain::DomainParticipant** (p. 1326) is enabled. Looking up builtin **dds::sub::DataReader** (p. 976) may implicitly register builtin transports due to creation of **dds::sub::DataReader** (p. 976) (see **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered). Therefore, if you are want to modify builtin transport properties, do so *before* using this operation.

Therefore the suggested sequence when looking up builtin DataReaders is:

- Create a disabled **dds::domain::DomainParticipant** (p. 1326).
- (optional) Modify builtin transport properties
- Call **dds::sub::builtin_subscriber()** (p. 671).
- Call **dds::sub::find()** (p. 672).
- Call **enable()** (p. 581) on the DomainParticipant.

The returned **dds::sub::DataReader** (p. 976) may be enabled or disabled.

If more than one **dds::sub::DataReader** (p. 976) is attached to the **dds::sub::Subscriber** (p. 2410), this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **dds::sub::DataReader** (p. 976) in one thread while another thread is simultaneously creating or destroying that **dds::sub::DataReader** (p. 976).

Template Parameters

Reader	The type of the reader. It can be **dds::sub::AnyDataReader** (p. 807), or an instantiation of dds::sub::DataReader<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))
BinIterator	The type of back-inserting iterator passed to this function

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) the **DataReader** (p. 976) belongs to
topic_name	Name of the **dds::topic::Topic** (p. 2474) associated with the **DataReader** (p. 976) that is to be looked up.
begin	A back-inserting iterator to the position in the destination container to insert the found **DataReader** (p. 976) into

Returns

The number of DataReaders that were found (either 0 or 1)

7.12.2.18 find() [3/6]

```
template<typename READER , typename T , typename FwdIterator >
uint32_t find (
            const  dds::sub::Subscriber & subscriber,
            const  dds::topic::TopicDescription< T > & topic_description,
            FwdIterator begin,
            uint32_t max_size )
```

This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching TopicDescription.

See also

find(const dds::sub::Subscriber& subscriber,RTI_STRING_IN topic_name,FwdIterator begin,uint32_t max_size) (p. 672)

7.12.2.19 find() [4/6]

```
template<typename READER , typename T , typename BinIterator >
uint32_t find (
            const  dds::sub::Subscriber & subscriber,
            const  dds::topic::TopicDescription< T > & topic_description,
            BinIterator begin )
```

This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching TopicDescription.

See also

uint32_t **find(const dds::sub::Subscriber& subscriber,RTI_STRING_IN topic_name,BinIterator begin)** (p. 673)

7.12.2.20 find() [5/6]

```
template<typename AnyDataReaderFwdIterator >
uint32_t dds::sub::find (
             const  dds::sub::Subscriber & subscriber,
             const  dds::sub::status::DataState & data_state,
             AnyDataReaderFwdIterator begin,
             uint32_t max_size )
```

Allows the application to access the AnyDataReaders that contain samples with the specified **dds::sub::status::DataState** (p. 1118).

Allows the application to access the **dds::sub::DataReader** (p. 976) objects that contain samples with the specified `sample_states`, `view_states` and `instance_states`.

If the application is outside a begin_access()/end_access() block, or if the **dds::core::policy::Presentation::access_scope** (p. 1947) of the **dds::sub::Subscriber** (p. 2410) is **dds::core::policy::PresentationAccessScopeKind_def::INSTANCE** (p. 1951) or **dds::core::policy::PresentationAccessScopeKind_def::TOPIC** (p. 1951), or if the **dds::core::policy::Presentation::ordered_access** (p. 1948) of the **dds::sub::Subscriber** (p. 2410) is false, the returned collection is a 'set' containing each **dds::sub::DataReader** (p. 976) at most once, in no specified order.

If the application is within a begin_access()/end_access() block, and the **PRESENTATION** (p. 540) policy of the **dds::sub::Subscriber** (p. 2410) is **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951) or **dds::core::policy::PresentationAccessScopeKind_def::HIGHEST_OFFERED** (p. 1951), and **dds::core::policy::Presentation::ordered_access** (p. 1948) in the **dds::sub::Subscriber** (p. 2410) is true, the returned collection is a 'list' of DataReaders where a **DataReader** (p. 976) may appear more than one time.

To retrieve the samples in the order they were published across DataWriters of the same group (**dds::pub::Publisher** (p. 1993) configured with **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951)), the application should **read()** (p. 661)/take() from each **DataReader** (p. 976) in the same order as it appears in the output sequence. The application will move to the next **DataReader** (p. 976) when the **read()** (p. 661)/take() operation does not return any data.

See also

Access to data samples (p. 62)

dds::sub::CoherentAccess::CoherentAccess() (p. 930)

dds::sub::CoherentAccess::end() (p. 931)

PRESENTATION (p. 540)

Template Parameters

AnyDataReaderFwdIterator	A forward iterator whose `value_type` is **dds::sub::AnyDataReader** (p. 807)

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) the DataReaders belong to
data_state	The DataState describing what kinds of samples the returned DataReaders must contain

Parameters

begin	A forward iterator to the position in the destination container where the DataReaders will be copied into. This container should be a container of AnyDataReaders, not typed DataReaders
max_size	The maximum number of DataReaders to return

Returns

The number of DataReaders that were found with a matching DataState

References **begin()**, **dds::sub::status::DataState::instance_state()**, **dds::sub::status::DataState::sample_state()**, and **dds::sub::status::DataState::view_state()**.

7.12.2.21 find() [6/6]

```
template<typename AnyDataReaderBackInsertIterator >
uint32_t dds::sub::find (
            const  dds::sub::Subscriber & subscriber,
            const  dds::sub::status::DataState & data_state,
            AnyDataReaderBackInsertIterator begin )
```

Allows the application to access the AnyDataReaders that contain samples with the specified **dds::sub::status::DataState** (p. 1118).

Allows the application to access the **dds::sub::DataReader** (p. 976) objects that contain samples with the specified `sample_states`, `view_states` and `instance_states`.

If the application is outside a begin_access()/end_access() block, or if the **dds::core::policy::Presentation::access_scope** (p. 1947) of the **dds::sub::Subscriber** (p. 2410) is **dds::core::policy::PresentationAccessScopeKind_def::INSTANCE** (p. 1951) or **dds::core::policy::PresentationAccessScopeKind_def::TOPIC** (p. 1951), or if the **dds::core::policy::Presentation::ordered_access** (p. 1948) of the **dds::sub::Subscriber** (p. 2410) is false, the returned collection is a 'set' containing each **dds::sub::DataReader** (p. 976) at most once, in no specified order.

If the application is within a begin_access()/end_access() block, and the **PRESENTATION** (p. 540) policy of the **dds::sub::Subscriber** (p. 2410) is **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951) or **dds::core::policy::PresentationAccessScopeKind_def::HIGHEST_OFFERED** (p. 1951), and **dds::core::policy::Presentation::ordered_access** (p. 1948) in the **dds::sub::Subscriber** (p. 2410) is true, the returned collection is a 'list' of DataReaders where a **DataReader** (p. 976) may appear more than one time.

To retrieve the samples in the order they were published across DataWriters of the same group (**dds::pub::Publisher** (p. 1993) configured with **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951)), the application should **read()** (p. 661)/take() from each **DataReader** (p. 976) in the same order as it appears in the output sequence. The application will move to the next **DataReader** (p. 976) when the **read()** (p. 661)/take() operation does not return any data.

See also

Access to data samples (p. 62)

dds::sub::CoherentAccess::CoherentAccess() (p. 930)

dds::sub::CoherentAccess::end() (p. 931)

PRESENTATION (p. 540)

Template Parameters

AnyDataReaderBackInsertIterator	A back-inserting iterator whose `value_type` is **dds::sub::AnyDataReader** (p. 807)

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) the DataReaders belong to
data_state	The DataState describing what kinds of samples the returned DataReaders must contain
begin	A back-inserting iterator to the position in the destination container to insert the found DataReaders into. This container should be a container of AnyDataReaders, not typed DataReaders

Returns

The number of DataReaders that were found with a matching DataState

References **begin()**, **dds::sub::status::DataState::instance_state()**, **dds::sub::status::DataState::sample_state()**, and **dds::sub::status::DataState::view_state()**.

7.12.2.22 move()

```
template<typename T >
LoanedSamples< T > move (
              LoanedSamples< T > & ls )
```

Creates a new **LoanedSamples** (p. 1663) instance by moving the contents of an existing one.

This function existed to provide move semantics before C++11, but is now deprecated in favor of std::move.

[DEPRECATED] Use std::move and the move constructor of **LoanedSamples** (p. 1663) instead. For example:

```
LoanedSamples<Foo> samples1 = reader.take();
LoanedSamples<Foo> samples2 = std::move(samples1);
```

Parameters

ls	The **LoanedSamples** (p. 1663) object that transfers its ownership of the contained samples into the returned object. After this call, `ls` is empty.

Returns

A new **LoanedSamples** (p. 1663) object, the new loan owner, with the same contents as `ls` had.

See also

LoanedSamples (p. 1663)

Referenced by **dds::sub::LoanedSamples< T >::move()**.

7.12.2.23 begin() [1/2]

```
template<typename T >
LoanedSamples< T >::iterator begin (
            LoanedSamples< T > & ls )
```

See also

LoanedSamples::begin() (p. 1667)

Referenced by **find()**, **dds::sub::DataReader< T >::find()**, **dds::sub::DataReader< T >::ignore()**, and **dds::sub::DataReader< T >::matched_publications()**.

7.12.2.24 begin() [2/2]

```
template<typename T >
LoanedSamples< T >::const_iterator begin (
           const  LoanedSamples< T > & ls )
```

See also

LoanedSamples::begin() (p. 1667)

7.12.2.25 end() [1/2]

```
template<typename T >
LoanedSamples< T >::iterator end (
            LoanedSamples< T > & ls )
```

See also

LoanedSamples::end() (p. 1668)

Referenced by **dds::sub::DataReader< T >::ignore()**, and **dds::sub::DataReader< T >::matched_publications()**.

7.12.2.26 end() [2/2]

```
template<typename T >
LoanedSamples< T >::const_iterator end (
             const  LoanedSamples< T > & ls )
```

See also

LoanedSamples::end() (p. 1668)

7.12.2.27 swap()

```
template<typename T >
void swap (
              LoanedSamples< T > & ls1,
              LoanedSamples< T > & ls2 ) throw ( )
```

See also

LoanedSamples::swap() (p. 1668)

7.13 dds::sub::cond Namespace Reference

Contains Conditions specific to DataReaders.

Classes

- class **QueryCondition**

  *<<**reference-type**>> (p. 351) Specialized **ReadCondition** (p. 2133) that allows applications to also specify a filter on the data available in a **dds::sub::DataReader** (p. 976)*
- class **ReadCondition**

  *<<**reference-type**>> (p. 351) Condition specifically dedicated to read operations and attached to one **dds::sub::DataReader** (p. 976).*

7.13.1 Detailed Description

Contains Conditions specific to DataReaders.

See also

dds::core::cond (p. 627)

7.14 dds::sub::qos Namespace Reference

Contains **DataReaderQos** (p. 1072) and **SubscriberQos** (p. 2423).

Classes

- class **DataReaderQos**

  <<***value-type***>> *(p. 350) Container of the QoS policies that a* ***dds::sub::DataReader*** *(p. 976) supports*
- class **SubscriberQos**

  <<***value-type***>> *(p. 350) Container of the QoS policies that a* ***dds::sub::Subscriber*** *(p. 2410) supports*

Functions

- std::string **to_string** (const **DataReaderQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::sub::qos::DataReaderQos*** *(p. 1072)*
- std::string **to_string** (const **DataReaderQos** &qos, const **DataReaderQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::sub::qos::DataReaderQos*** *(p. 1072)*
- std::string **to_string** (const **DataReaderQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::sub::qos::DataReaderQos*** *(p. 1072)*
- std::ostream & **operator<<** (std::ostream &out, const rti::sub::qos::DataReaderQos &qos)

  <<***extension***>> *(p. 354) Prints a* ***dds::sub::qos::DataReaderQos*** *(p. 1072) to an output stream.*
- std::string **to_string** (const **SubscriberQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::sub::qos::SubscriberQos*** *(p. 2423)*
- std::string **to_string** (const **SubscriberQos** &qos, const **SubscriberQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::sub::qos::SubscriberQos*** *(p. 2423)*
- std::string **to_string** (const **SubscriberQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::sub::qos::SubscriberQos*** *(p. 2423)*
- std::ostream & **operator<<** (std::ostream &out, const rti::sub::qos::SubscriberQos &qos)

  <<***extension***>> *(p. 354) Prints a* ***dds::sub::qos::SubscriberQos*** *(p. 2423) to an output stream.*

7.14.1 Detailed Description

Contains **DataReaderQos** (p. 1072) and **SubscriberQos** (p. 2423).

7.14.2 Function Documentation

7.14.2.1 to_string() [1/6]

```
std::string to_string (
            const  DataReaderQos & qos,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(*p. *354)* Obtains a string representation of the **dds::sub::qos::DataReaderQos** (p. 1072)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
DataReaderQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for DataReaderQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
DataReaderQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

7.14.2.2 to_string() [2/6]

```
std::string to_string (
            const  DataReaderQos & qos,
            const  DataReaderQos & base,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::sub::qos::DataReaderQos** (p. 1072)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

7.14.2.3 to_string() [3/6]

```
std::string to_string (
             const  DataReaderQos & qos,
             const  rti::core::qos_print_all_t & qos_print_all,
             const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::sub::qos::DataReaderQos** (p. 1072)

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
```
to_string(qos, rti::core::qos_print_all);
```

Returns

The string representation of the qos

7.14.2.4 operator<<() [1/2]

```
std::ostream & operator<< (
             std::ostream & out,
             const rti::sub::qos::DataReaderQos & qos )  [inline]
```

<<***extension***>> *(p. 354)* Prints a **dds::sub::qos::DataReaderQos** (p. 1072) to an output stream.

7.14.2.5 to_string() [4/6]

```
std::string to_string (
             const  SubscriberQos & qos,
             const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::sub::qos::SubscriberQos** (p. 2423)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
SubscriberQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for SubscriberQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
SubscriberQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

7.14.2.6 to_string() [5/6]

```
std::string to_string (
            const  SubscriberQos & qos,
            const  SubscriberQos & base,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::sub::qos::SubscriberQos** (p. 2423)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

7.14.2.7 to_string() [6/6]

```
std::string to_string (
             const  SubscriberQos & qos,
             const  rti::core::qos_print_all_t & qos_print_all,
             const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::sub::qos::SubscriberQos** (p. 2423)

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
```
to_string(qos, rti::core::qos_print_all);
```

Returns

The string representation of the qos

7.14.2.8 operator<<() [2/2]

```
std::ostream & operator<< (
             std::ostream & out,
             const rti::sub::qos::SubscriberQos & qos )  [inline]
```

<<***extension***>> *(p. 354)* Prints a **dds::sub::qos::SubscriberQos** (p. 2423) to an output stream.

7.15 dds::sub::status Namespace Reference

Contains **DataState** (p. 1118).

Classes

- class **DataState**

  *The **DataState** (p. 1118) class describes the state of a sample and includes the information about the sample's **InstanceState** (p. 1615), **ViewState** (p. 2614), and **SampleState** (p. 2316).*
- class **InstanceState**

  *Indicates if the samples are from a live **dds::pub::DataWriter** (p. 1138) or not.*
- class **SampleState**

  Indicates whether or not a sample has ever been read.
- class **ViewState**

  Indicates whether or not an instance is new.

Functions

- std::ostream & **operator<<** (std::ostream &os, const **SampleState** &s)

  Prints a sample state as a readable string.
- std::ostream & **operator<<** (std::ostream &os, const **ViewState** &s)

  Prints a view state as a readable string.
- std::ostream & **operator<<** (std::ostream &os, const **InstanceState** &s)

  Prints an instance state as a readable string.
- std::ostream & **operator<<** (std::ostream &os, const **DataState** &s)

  Prints a data state as a readable string.

7.15.1 Detailed Description

Contains **DataState** (p. 1118).

7.15.2 Function Documentation

7.15.2.1 operator<<() [1/4]

```
std::ostream & operator<< (
            std::ostream & os,
            const  SampleState & s )
```

Prints a sample state as a readable string.

7.15.2.2 operator<<() [2/4]

```
std::ostream & operator<< (
            std::ostream & os,
            const  ViewState & s )
```

Prints a view state as a readable string.

7.15.2.3 operator<<() [3/4]

```
std::ostream & operator<< (
            std::ostream & os,
            const  InstanceState & s )
```

Prints an instance state as a readable string.

7.15.2.4 operator<<() [4/4]

```
std::ostream & operator<< (
            std::ostream & os,
            const  DataState & s )
```

Prints a data state as a readable string.

7.16 dds::topic Namespace Reference

Contains topic related classes and functions, the built-in topics and topic traits used by IDL-generated types.

Namespaces

- namespace **qos**

  *Contains **TopicQos** (p. 2509).*

Classes

- class **AnyTopic**

  *<<**reference-type**>> (p. 351) This class provides an non-template holder for representing a **Topic** (p. 2474) of any type*
- class **BuiltinTopicKey**

  The key of the built-in topics.
- class **ContentFilteredTopic**

  *<<**reference-type**>> (p. 351) Specialization of **TopicDescription** (p. 2503) that allows for content-based subscriptions.*
- class **Filter**

  *Defines the filter to create a **ContentFilteredTopic** (p. 954).*
- struct **is_topic_type**

  *Trait that indicates if a type is suitable to be the type of a **dds::topic::Topic** (p. 2474).*
- class **NoOpTopicListener**

  *A convenience implementation of **TopicListener** (p. 2508) where all methods are overridden to do nothing.*
- class **ParticipantBuiltinTopicData**

  *Entry created when a **dds::domain::DomainParticipant** (p. 1326) is discovered.*
- class **PublicationBuiltinTopicData**

  *Entry created when a **dds::pub::DataWriter** (p. 1138) is discovered in association with its **dds::pub::Publisher** (p. 1993).*
- class **SubscriptionBuiltinTopicData**

  *Entry created when a **dds::sub::DataReader** (p. 976) is discovered in association with its **dds::sub::Subscriber** (p. 2410).*
- class **Topic**

  *<<**reference-type**>> (p. 351) **Topic** (p. 2474) is the most basic description of the data to be published and subscribed.*
- struct **topic_type_name**

  Provides the name of a topic-type.
- struct **topic_type_support**

  Provides convenience operations for a topic-type.
- class **TopicBuiltinTopicData**

  *Entry created when a **dds::topic::Topic** (p. 2474) object is discovered.*
- class **TopicDescription**

  *Abstract base class of **Topic** (p. 2474) and **ContentFilteredTopic** (p. 954).*
- class **TopicInstance**

  Encapsulates a sample and its associated instance handle.
- class **TopicListener**

  *The listener to notify status changes for a **dds::topic::Topic** (p. 2474).*

Functions

- template<typename T >
  Topic< T > **get** (const **AnyTopic** &any_topic)

  *Same as **AnyTopic::get()** (p. 827)*
- template<typename FwdIterator >
  int32_t **discover_any_topic** (const **dds::domain::DomainParticipant** &participant, FwdIterator begin, int32_t max_size)

  Retrieves the topics discovered in a DomainParticipant.
- template<typename BinIterator >
  int32_t **discover_any_topic** (const **dds::domain::DomainParticipant** &participant, BinIterator begin)

Retrieves the topics discovered in a DomainParticipant.

- **dds::topic::TopicBuiltinTopicData discover_topic_data** (const **dds::domain::DomainParticipant** &participant, const **dds::core::InstanceHandle** &topic_handle)

  Gets the information about a discovered topic.

- template<typename FwdIterator >
  int32_t **discover_topic_data** (const **dds::domain::DomainParticipant** &participant, FwdIterator begin, const **dds::core::InstanceHandleSeq** &handles)

  Gets the information about several topics.

- template<typename FwdIterator >
  int32_t **discover_topic_data** (const **dds::domain::DomainParticipant** &participant, FwdIterator begin, int32_t max_size)

  Gets the information about all topics.

- template<typename BinIterator >
  int32_t **discover_topic_data** (const **dds::domain::DomainParticipant** &participant, BinIterator begin)

  Gets the information about all topics.

- void **ignore** (**dds::domain::DomainParticipant** &participant, const **dds::core::InstanceHandle** &handle)

  Instructs a DomainParticipant to ignore a topic.

- template<typename FwdIterator >
  void **ignore** (**dds::domain::DomainParticipant** &participant, FwdIterator begin, FwdIterator end)

  Ignores a range of topics.

- template<typename TOPIC >
  TOPIC **find** (const **dds::domain::DomainParticipant** &participant, const std::string &topic_name)

  *Looks up a **Topic** (p. 2474) from a DomainParticipant.*

- std::string **participant_topic_name** ()

  Participant builtin topic name.

- std::string **topic_topic_name** ()

  ***Topic** (p. 2474) topic name.*

- std::string **publication_topic_name** ()

  Publication topic name.

- std::string **subscription_topic_name** ()

  Subscription topic name.

7.16.1 Detailed Description

Contains topic related classes and functions, the built-in topics and topic traits used by IDL-generated types.

7.16.2 Function Documentation

7.16.2.1 get()

```
template<typename T >
Topic< T > get (
            const  AnyTopic & any_topic )
```

Same as **AnyTopic::get()** (p. 827)

7.16.2.2 discover_any_topic() [1/2]

```
template<typename FwdIterator >
int32_t discover_any_topic (
            const  dds::domain::DomainParticipant & participant,
            FwdIterator begin,
            int32_t max_size )
```

Retrieves the topics discovered in a DomainParticipant.

Template Parameters

FwdIterator	A forward iterator whose value type is **dds::core::InstanceHandle** (p. 1612)

Parameters

participant	The DomainParticipant where to look up the discovered topics
begin	The beginning of the range where to insert the InstanceHandles that correspond to the discovered topics
max_size	The maximum number of topics to insert or **dds::core::LENGTH_UNLIMITED** (p. 438)

Referenced by **dds::topic::Topic< T >::discover_any_topic()**.

7.16.2.3 discover_any_topic() [2/2]

```
template<typename BinIterator >
int32_t discover_any_topic (
            const  dds::domain::DomainParticipant & participant,
            BinIterator begin )
```

Retrieves the topics discovered in a DomainParticipant.

This is equivalent to discover_any_topic(participant, begin, dds::core::LENGTH_UNLIMITED)

7.16.2.4 discover_topic_data() [1/4]

```
dds::topic::TopicBuiltinTopicData dds::topic::discover_topic_data (
            const  dds::domain::DomainParticipant & participant,
            const  dds::core::InstanceHandle & topic_handle )
```

Gets the information about a discovered topic.

Returns **dds::topic::TopicBuiltinTopicData** (p. 2493) for the specified **dds::topic::Topic** (p. 2474).

This operation retrieves information on a **dds::topic::Topic** (p. 2474) that has been discovered by the local Participant and must not have been "ignored" by means of the **dds::topic::ignore()** (p. 693) operation.

The `topic_handle` must correspond to such a topic. Otherwise, the operation will fail with **dds::core::PreconditionNotMetError** (p. 1941).

This call is not supported for remote topics. If a remote `topic_handle` is used, the operation will fail with **dds::core::UnsupportedError** (p. 2590).

Use the operation **dds::topic::discover_any_topic()** (p. 690) to find the topics that are currently discovered.

Parameters

participant	The DomainParticipant where to look up the topic information
topic_handle	<<***in***>> *(p. 355)* **dds::core::InstanceHandle** (p. 1612) of **dds::topic::Topic** (p. 2474).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::PreconditionNotMetError** (p. 1941) or **dds::core::NotEnabledError** (p. 1869)

See also

dds::topic::TopicBuiltinTopicData (p. 2493)

dds::topic::discover_any_topic() (p. 690)

Referenced by **dds::topic::Topic< T >::discover_topic_data()**.

7.16.2.5 discover_topic_data() [2/4]

```
template<typename FwdIterator >
int32_t discover_topic_data (
             const  dds::domain::DomainParticipant & participant,
             FwdIterator begin,
             const  dds::core::InstanceHandleSeq & handles )
```

Gets the information about several topics.

7.16.2.6 discover_topic_data() [3/4]

```
template<typename FwdIterator >
int32_t discover_topic_data (
             const  dds::domain::DomainParticipant & participant,
             FwdIterator begin,
             int32_t max_size )
```

Gets the information about all topics.

Template Parameters

A	forward iterator whose value type is **dds::topic::TopicBuiltinTopicData** (p. 2493)

Parameters

participant	The DomainParticipant where to look up all the topics
begin	The beginning of a range where to copy the topic data
max_size	The maximum number of items to copy or **dds::core::LENGTH_UNLIMITED** (p. 438)

7.16.2.7 discover_topic_data() [4/4]

```
template<typename BinIterator >
int32_t discover_topic_data (
             const  dds::domain::DomainParticipant & participant,
             BinIterator begin )
```

Gets the information about all topics.

7.16.2.8 ignore() [1/2]

```
void ignore (
              dds::domain::DomainParticipant & participant,
             const  dds::core::InstanceHandle & handle )
```

Instructs a DomainParticipant to ignore a topic.

Instructs RTI Connext to locally ignore a **dds::topic::Topic** (p. 2474).

This means it will locally ignore any publication, or subscription to the **dds::topic::Topic** (p. 2474).

There is no way to reverse this operation.

This operation can be used to save local resources when the application knows that it will never publish or subscribe to data under certain topics.

The **dds::topic::Topic** (p. 2474) to ignore is identified by the `handle` argument. This is the handle of a **dds::topic::Topic** (p. 2474) that appears in the **dds::sub::SampleInfo** (p. 2285) retrieved when reading data samples from the built-in **dds::sub::DataReader** (p. 976) for the **dds::topic::Topic** (p. 2474).

Parameters

participant	The DomainParticipant where to ignore the topic
handle	<<***in***>> *(p. 355)* Handle of the **dds::topic::Topic** (p. 2474) to be ignored.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::OutOfResourcesError** (p. 1898) or **dds::core::NotEnabledError** (p. 1869)

See also

dds::topic::TopicBuiltinTopicData (p. 2493)

dds::topic::topic_topic_name() (p. 442)

dds::sub::builtin_subscriber (p. 671)

Referenced by **dds::topic::Topic< T >::ignore()**.

7.16.2.9 ignore() [2/2]

```
template<typename FwdIterator >
void ignore (
              dds::domain::DomainParticipant & participant,
             FwdIterator begin,
             FwdIterator end )
```

Ignores a range of topics.

See also

ignore(dds::domain::DomainParticipant&,const dds::core::InstanceHandle&) (p. 693)

7.16.2.10 find()

```
template<typename TOPIC >
TOPIC find (
             const  dds::domain::DomainParticipant & participant,
             const std::string & topic_name )
```

Looks up a **Topic** (p. 2474) from a DomainParticipant.

Template Parameters

TOPIC	The topic to find; it can be one of the following: • A typed **TopicDescription** (p. 2503), such as `dds::topic::TopicDescription<Foo>`, • A typed **Topic** (p. 2474), such as `dds::topic::Topic<Foo>` • A typed **ContentFilteredTopic** (p. 954), such as `dds::topic::ContentFilteredTopic<Foo>` • **`dds::topic::AnyTopic`** (p. 825)

Parameters

participant	The DomainParticipant that contains the **Topic** (p. 2474)
topic_name	The topic name

Returns

A reference to an existing **Topic** (p. 2474) in this participant or an empty reference (i.e. equals to **dds::core::null** (p. 438)) if it doesn't exist.

Exceptions

dds::core::InvalidDowncastError *(p. 1620)*	If the concrete type of the topic description identified by `topic_name` is not `TOPIC`. For example, if the topic_name identifies a topic of type `Bar`, but the template parameter was dds::topic::Topic<Foo>.

The participant doesn't need to be enabled. The returned topic may be enabled or disabled.

7.17 dds::topic::qos Namespace Reference

Contains **TopicQos** (p. 2509).

Classes

- class **TopicQos**

  *<<**value-type**>> (p. 350) Container of the QoS policies that a **dds::topic::Topic** (p. 2474) supports*

Functions

- std::string **to_string** (const **TopicQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrint↩ Format**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::topic::qos::TopicQos*** *(p. 2509)*
- std::string **to_string** (const **TopicQos** &qos, const **TopicQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::topic::qos::TopicQos*** *(p. 2509)*
- std::string **to_string** (const **TopicQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti::core↩ ::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::topic::qos::TopicQos*** *(p. 2509)*
- std::ostream & **operator**<< (std::ostream &out, const rti::topic::qos::TopicQos &qos)

  <<***extension***>> *(p. 354) Prints a* ***dds::topic::qos::TopicQos*** *(p. 2509) to an output stream.*

7.17.1 Detailed Description

Contains **TopicQos** (p. 2509).

7.17.2 Function Documentation

7.17.2.1 to_string() [1/3]

```
std::string to_string (
            const  TopicQos & qos,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::topic::qos::TopicQos** (p. 2509)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
TopicQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for TopicQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
```

```
TopicQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

7.17.2.2 to_string() [2/3]

```
std::string to_string (
            const  TopicQos & qos,
            const  TopicQos & base,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::topic::qos::TopicQos** (p. 2509)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

7.17.2.3 to_string() [3/3]

```
std::string to_string (
            const  TopicQos & qos,
            const  rti::core::qos_print_all_t & qos_print_all,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::topic::qos::TopicQos** (p. 2509)

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
```
to_string(qos, rti::core::qos_print_all);
```

Returns

The string representation of the qos

7.17.2.4 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const rti::topic::qos::TopicQos & qos )  [inline]
```

<<***extension***>> *(p. 354)* Prints a **dds::topic::qos::TopicQos** (p. 2509) to an output stream.

7.18 rti Namespace Reference

<<***extension***>> *(p. 354)* The namespace that contains the extension types and functions to the DDS standard.

Namespaces

- namespace **all**
  <<***extension***>> *(p. 354) A single namespace where all symbols are included.*
- namespace **config**
  <<***extension***>> *(p. 354) Logging configuration and version information.*
- namespace **core**
  <<***extension***>> *(p. 354) Extensions to* ***dds::core*** *(p. 616)*
- namespace **domain**
  <<***extension***>> *(p. 354) Extensions to* ***dds::domain*** *(p. 634)*
- namespace **flat**
  <<***extension***>> *(p. 354) Support for* ***FlatData Topic-Types*** *(p. 419)*
- namespace **pub**
  <<***extension***>> *(p. 354) Extensions to* ***dds::pub*** *(p. 645)*
- namespace **sub**
  <<***extension***>> *(p. 354) Extensions to* ***dds::sub*** *(p. 658)*
- namespace **topic**
  <<***extension***>> *(p. 354) Extensions to* ***dds::topic*** *(p. 688)*
- namespace **util**
  <<***extension***>> *(p. 354) Contains general-purpose utilities*

7.18.1 Detailed Description

<<***extension***>> *(p. 354)* The namespace that contains the extension types and functions to the DDS standard.

Copyright (c) 2013, Real-Time Innovations, Inc. All rights reserved.

No duplications, whole or partial, manual or electronic, may be made without express written permission. Any such copies, or revisions thereof, must display this notice unaltered. This code contains trade secrets of Real-Time Innovations, Inc.

Copyright (c) 2019, Real-Time Innovations, Inc. All rights reserved.

No duplications, whole or partial, manual or electronic, may be made without express written permission. Any such copies, or revisions thereof, must display this notice unaltered. This code contains trade secrets of Real-Time Innovations, Inc.

Copyright (c) 2013, Real-Time Innovations, Inc. All rights reserved.

No duplications, whole or partial, manual or electronic, may be made without express written permission. Any such copies, or revisions thereof, must display this notice unaltered. This code contains trade secrets of Real-Time Innovations, Inc.

The rti namespace mirrors the structure of the dds namespace with the same sub-namespaces and two more: **rti↩ ::config** (p. 699) and **rti::util** (p. 777).

7.19 rti::all Namespace Reference

<<***extension***>> *(p. 354)* A single namespace where all symbols are included.

7.19.1 Detailed Description

<<***extension***>> *(p. 354)* A single namespace where all symbols are included.

This is similar to **dds::all** (p. 615) but it also includes the extension symbols. This namespace is defined in "rti/rti.hpp".

See also

dds::all (p. 615)

7.20 rti::config Namespace Reference

<<***extension***>> *(p. 354)* Logging configuration and version information.

Classes

- class **LibraryVersion**

  <<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) *The version of a single library shipped as part of an RTI Connext distribution.*
- struct **LogCategory_def**

  The definition of the ***dds::core::safe_enum*** (p. *2265*) *LogCategory.*
- class **Logger**

  The singleton type used to configure RTI Connext logging.
- struct **LogLevel_def**

  The definition of the ***dds::core::safe_enum*** (p. *2265*) *LogLevel;.*
- struct **LogMessage**

  A log message's input parameter to an output handler, including the text and additional information.
- struct **PrintFormat_def**

  The definition of the ***dds::core::safe_enum*** (p. *2265*) *PrintFormat.*
- class **ScopedLoggerVerbosity**

  Changes the logger verbosity temporarily during the scope of a variable.
- struct **Verbosity_def**

  The definition of the ***dds::core::safe_enum*** (p. *2265*) *Verbosity.*

Typedefs

- typedef **dds::core::safe_enum**< **LogLevel_def** > **LogLevel**

  The log level at which RTI Connext diagnostic information is logged.
- typedef **dds::core::safe_enum**< **Verbosity_def** > **Verbosity**

  The verbosities at which RTI Connext diagnostic information is logged.
- typedef **dds::core::safe_enum**< **LogCategory_def** > **LogCategory**

  Categories of logged messages.
- typedef **dds::core::safe_enum**< **PrintFormat_def** > **PrintFormat**

  The format used to output RTI Connext diagnostic information.

Enumerations

- enum class **SyslogLevel** {
  SyslogLevel::emergency ,
  SyslogLevel::alert ,
  SyslogLevel::critical ,
  SyslogLevel::error ,
  SyslogLevel::warning ,
  SyslogLevel::notice ,
  SyslogLevel::informational ,
  SyslogLevel::debug }

- enum class **SyslogVerbosity** {
  SyslogVerbosity::silent ,
  SyslogVerbosity::emergency ,
  SyslogVerbosity::alert ,
  SyslogVerbosity::critical ,
  SyslogVerbosity::error ,
  SyslogVerbosity::warning ,
  SyslogVerbosity::notice ,
  SyslogVerbosity::informational ,
  SyslogVerbosity::debug }
- enum class **LogFacility** {
  LogFacility::user ,
  LogFacility::security_event ,
  LogFacility::service ,
  LogFacility::middleware }

Functions

- XMQCPP2DllExport **rti::core::ProductVersion request_reply_api_version** ()

  Get the version of the C++ API library.
- XMQCPP2DllExport std::string **request_reply_build_number** ()

  Get the build number of the C++ API library.
- void **enable_automatic_tss_cleanup** (bool enable)

  Enable automatic thread-specific storage (TSS) cleanup on platforms that support it.
- **LibraryVersion core_version** ()

  Get the version of the core library.
- std::string **core_build_number** ()

  Get the build number of the core library.
- **LibraryVersion c_api_version** ()

  Get the version of the C API library.
- std::string **c_build_number** ()

  Get the build number of the C API library.
- **LibraryVersion cpp_api_version** ()

  Get the version of the C++ API library.
- std::string **cpp_build_number** ()

  Get the build number of the C++ API library.
- **rti::core::ProductVersion product_version** ()

  Get the RTI Connext product version.

7.20.1 Detailed Description

<<***extension***>> *(*p. *354)* Logging configuration and version information.

7.21 rti::core Namespace Reference

<<***extension***>> *(*p. *354)* Extensions to **dds::core** (p. 616)

Namespaces

- namespace **builtin_profiles**

  *<<**extension**>> (p. 354) Contains the names of the built-in profiles*
- namespace **xtypes**

  *<<**extension**>> (p. 354) Extensions to **dds::core::xtypes** (p. 631)*

Classes

- class **AllocationSettings**

  *<<**extension**>> (p. 354) Resource allocation settings*
- class **bounded_sequence**

  *<<**value-type**>> (p. 350) A bounded sequence of elements*
- class **ChannelSettings**

  *<<**extension**>> (p. 354) Configures the properties of a channel in **rti::core::policy::MultiChannel** (p. 1749)*
- class **CoherentSetInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A CoherentSampleInfo provides information about the coherent set associated with a sample.*
- class **CompressionIdMask**

  *<<**extension**>> (p. 354) Mask that specifies which built-in compression method to used.*
- class **CompressionSettings**

  *<<**extension**>> (p. 354) Compression Settings*
- class **ContentFilterProperty**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Provides all the required information to enable content filtering.*
- class **Cookie**

  Unique identifier for a written data sample in the form of a sequence of bytes.
- class **DataReaderResourceLimitsInstanceReplacementSettings**

  *<<**extension**>> (p. 354) How instances are replaced in the DataReader queue when resource limits are reached.*
- class **DataWriterShmemRefTransferModeSettings**

  *<<**extension**>> (p. 354) Configures aspects of the shared memory reference transfer mode related to a DataWriter*
- class **EndpointGroup**

  *<<**extension**>> (p. 354) Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.*
- class **Guid**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Class for GUID (Global Unique Identifier) representation*
- class **ListenerBinder**

  *<<**reference-type**>> (p. 351) Automatically manages the association of an Entity and a **Listener** (p. 1637)*
- class **Locator**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.*
- class **LocatorFilterElement**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Specifies the configuration of an individual channel within a Multi↩ Channel DataWriter.*
- struct **LocatorKind_def**

  *The definition of the **dds::core::safe_enum** (p. 2265) LocatorKind.*
- class **LongDouble**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Encapsulates an IDL `long double`*

- class **MonitoringDedicatedParticipantSettings**

  *<<**extension**>> (p. 354) Configures the usage of a dedicated **dds::domain::DomainParticipant** (p. 1326) to distribute the RTI Connext application telemetry data.*
- class **MonitoringDistributionSettings**

  *<<**extension**>> (p. 354) Configures the distribution of telemetry data.*
- class **MonitoringEventDistributionSettings**

  *<<**extension**>> (p. 354) Configures the distribution of event metrics.*
- class **MonitoringLoggingDistributionSettings**

  *<<**extension**>> (p. 354) Configures the distribution of log messages.*
- class **MonitoringLoggingForwardingSettings**

  *<<**extension**>> (p. 354) Configures the forwarding levels of log messages for the different **rti::config::LogFacility** (p. 459).*
- class **MonitoringMetricSelection**

  *<<**extension**>> (p. 354) Configures event and periodic metrics collection and distribution for a specific set of observable resources.*
- class **MonitoringPeriodicDistributionSettings**

  *<<**extension**>> (p. 354) Configures the distribution of periodic metrics.*
- class **MonitoringTelemetryData**

  *<<**extension**>> (p. 354) Configures the telemetry data that will be distributed.*
- class **optional_value**

  *<<**extension**>> (p. 354) Represents a value that can be initialized or not*
- class **PersistentStorageSettings**

  *<<**extension**>> (p. 354) Configures the persistent storage settings for durable writer history and durable reader state.*
- class **pointer**
- class **ProductVersion**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Represents the current version of RTI Connext*
- class **ProtocolVersion**

  *<<**extension**>> (p. 354) Represents the current version of RTI Connext*
- struct **qos_print_all_t**

  A tag type that selects the to_string overload that prints all the values of a Qos object.
- class **QosPrintFormat**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A collection of attributes used to configure how QoS will be formatted when converted to strings.*
- class **QosProviderParams**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Configure options that control the way that XML documents containing QoS profiles are loaded by a **dds::core::QosProvider** (p. 2025).*
- class **Result**

  A result from an operation that doesn't throw exceptions containing a return code and (if successful) a value.
- class **RtpsReliableReaderProtocol**

  *<<**extension**>> (p. 354) Configures aspects of the RTPS protocol related to a reliable DataReader*
- class **RtpsWellKnownPorts**

  *<<**extension**>> (p. 354) Configures the mapping of the RTPS well-known ports*
- class **SampleFlag**

  *<<**extension**>> (p. 354) A set of flags that can be associated with a sample.*
- class **SampleIdentity**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A **SampleIdentity** (p. 2282) defines a pair (Virtual Writer **Guid** (p. 1596), **SequenceNumber** (p. 2335)) that uniquely identifies a sample within a DDS domain and a Topic.*
- class **SequenceNumber**

<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A class representing the DDS 64-bit Sequence Number

- struct **ServiceRequestId_def**

  The definition of the rti::core::safe_enum ServiceRequestId.

- class **ThreadSettings**

  <<**extension**>> (p. 354) *The properties of a thread of execution.*

- struct **ThreadSettingsCpuRotationKind_def**

  Determines how **rti::core::ThreadSettings::cpu_list** (p. 2451) *affects processor affinity for thread-related QoS policies that apply to multiple threads.*

- class **ThreadSettingsKindMask**

  <<**extension**>> (p. 354) *A collection of flags used to configure threads of execution*

- struct **TransportClassId_def**

  The definition of the **dds::core::safe_enum** (p. 2265) *TransportClassId.*

- class **TransportInfo**

  <<**extension**>> (p. 354) <<**value-type**>> (p. 350) *Contains the class id and message max size of an installed transport*

- class **TransportMulticastSettings**

  <<**extension**>> (p. 354) *Represents a list of multicast locators*

- class **TransportUnicastSettings**

  <<**extension**>> (p. 354) *Represents a list of unicast locators*

- class **UnregisterThreadOnExit**

  <<**extension**>> (p. 354) *Utility that calls* **rti::core::unregister_thread** (p. 437) *when leaving scope*

- class **VendorId**

  <<**extension**>> (p. 354) *Represents the vendor of the service implementing the RTPS protocol.*

Typedefs

- typedef **dds::core::vector**< **EndpointGroup** > **EndpointGroupSeq**

  A sequence of **rti::core::EndpointGroup** (p. 1515).

- typedef **dds::core::safe_enum**< **LocatorKind_def** > **LocatorKind**

  Safe Enumeration (p. 429) *of* **LocatorKind_def** (p. 1681)

- typedef std::vector< **TransportUnicastSettings** > **TransportUnicastSettingsSeq**

  <<**extension**>> (p. 354) *A sequence of* **TransportUnicastSettings** (p. 2559)

- typedef std::vector< **TransportMulticastSettings** > **TransportMulticastSettingsSeq**

  A sequence of **TransportMulticastSettings** (p. 2549).

- typedef std::vector< **ChannelSettings** > **ChannelSettingsSeq**

  <<**extension**>> (p. 354) *A sequence of* **ChannelSettings** (p. 921)

- typedef std::vector< **MonitoringMetricSelection** > **MonitoringMetricSelectionSeq**

  <<**extension**>> (p. 354) *A sequence of* **rti::core::MonitoringMetricSelection** (p. 1736).

- typedef **dds::core::safe_enum**< **ServiceRequestId_def** > **ServiceRequestId**

  Safe Enumeration (p. 429) *of* **ServiceRequestId_def** (p. 2361)

- typedef **dds::core::safe_enum**< **ThreadSettingsCpuRotationKind_def** > **ThreadSettingsCpuRotation↩ Kind**

  Safe Enumeration (p. 429) *of* **ThreadSettingsCpuRotationKind_def** (p. 2453)

- typedef **dds::core::safe_enum**< **TransportClassId_def** > **TransportClassId**

  Safe Enumeration (p. 429) *of* **TransportClassId_def** (p. 2540)

- typedef **dds::core::vector**< **TransportInfo** > **TransportInfoSeq**

  A sequence of **TransportInfo** (p. 2541).

Functions

- template<typename T , size_t N>
  void **fill_array** (dds::core::array< T, N > &array, const T &value)
- template<typename FinalType , typename ArrayType , size_t N, size_t M>
  void **fill_array** (dds::core::array< dds::core::array< ArrayType, N >, M > &multi_array, const FinalType &value)
- template<typename CType , typename T , size_t N, typename InitFunc >
  void **initialize_native_array** (dds::core::array< T, N > &array, InitFunc init_func)
- template<typename T , size_t N>
  std::ostream & **operator**<< (std::ostream &out, const **bounded_sequence**< T, N > &v)

  Print a vector applying << to all of its elements.
- template<typename Entity , typename **Listener** >
  ListenerBinder< Entity, **Listener** > **bind_listener** (Entity entity, **Listener** *the_listener, **dds::core::status::StatusMask** mask)

  *Sets the listener and creates a **ListenerBinder** (p. 1641) that automatically resets it when all references go out of scope.*
- template<typename Entity , typename **Listener** >
  ListenerBinder< Entity, **Listener** > **bind_listener** (Entity entity, **Listener** *the_listener)

  *Sets the listener and creates a **ListenerBinder** (p. 1641) that automatically resets it when all references go out of scope.*
- template<typename Entity , typename **Listener** >
  ListenerBinder< Entity, **Listener** > **bind_and_manage_listener** (Entity entity, **Listener** *the_listener, **dds::core::status::StatusMask** mask)

  *Sets the listener and creates a **ListenerBinder** (p. 1641) that automatically resets and deletes it when all references go out of scope.*
- template<typename Entity , typename **Listener** >
  ListenerBinder< Entity, **Listener** > **bind_and_manage_listener** (Entity entity, **Listener** *the_listener)

  *Sets the listener and creates a **ListenerBinder** (p. 1641) that automatically resets and deletes it when all references go out of scope.*
- template<typename T >
  std::ostream & **operator**<< (std::ostream &out, const **pointer**< T > &ptr)
- **QosProviderParams** **default_qos_provider_params** ()

  *<<**extension**>> (p. 354) Get the **rti::core::QosProviderParams** (p. 2049) for the default QosProvider*
- void **default_qos_provider_params** (const **rti::core::QosProviderParams** &params)

  *<<**extension**>> (p. 354) Set the **rti::core::QosProviderParams** (p. 2049) for the default QosProvider*
- void **unregister_thread** ()

  *<<**extension**>> (p. 354) Allows you to release thread-specific resources kept by RTI Connext on platforms that do not support automatic thread-specific storage (TSS) cleanup.*

Variables

- constexpr int32_t **length_auto** = -2

  A special value indicating an auto quantity.
- const int32_t **PUBLICATION_PRIORITY_UNDEFINED**

  *Initializer value for **rti::core::policy::PublishMode::priority** (p. 2017) and/or **rti::core::ChannelSettings::priority** (p. 924).*
- const int32_t **PUBLICATION_PRIORITY_AUTOMATIC**

  *Constant value for **rti::core::policy::PublishMode::priority** (p. 2017) and/or **rti::core::ChannelSettings::priority** (p. 924).*
- const **qos_print_all_t** **qos_print_all**

  Sentinel value that selects the to_string overload that prints all of the values of a Qos object.

7.21.1 Detailed Description

<<***extension***>> *(p. 354)* Extensions to **dds::core** (p. 616)

7.21.2 Typedef Documentation

7.21.2.1 LocatorKind

```
typedef dds::core::safe_enum< LocatorKind_def> rti::core::LocatorKind
```

Safe Enumeration (p. 429) of **LocatorKind_def** (p. 1681)

See also

LocatorKind_def (p. 1681)

7.21.2.2 MonitoringMetricSelectionSeq

```
typedef std::vector< MonitoringMetricSelection> rti::core::MonitoringMetricSelectionSeq
```

<<***extension***>> *(p. 354)* A sequence of **rti::core::MonitoringMetricSelection** (p. 1736).

7.21.2.3 ThreadSettingsCpuRotationKind

```
typedef dds::core::safe_enum< ThreadSettingsCpuRotationKind_def> rti::core::ThreadSettingsCpu↩
RotationKind
```

Safe Enumeration (p. 429) of **ThreadSettingsCpuRotationKind_def** (p. 2453)

See also

ThreadSettingsCpuRotationKind_def (p. 2453)

7.21.2.4 TransportClassId

```
typedef dds::core::safe_enum< TransportClassId_def> rti::core::TransportClassId
```

Safe Enumeration (p. 429) of **TransportClassId_def** (p. 2540)

See also

TransportClassId_def (p. 2540)

7.21.2.5 TransportInfoSeq

```
typedef dds::core::vector< TransportInfo> rti::core::TransportInfoSeq
```

A sequence of **TransportInfo** (p. 2541).

7.21.3 Function Documentation

7.21.3.1 fill_array() [1/2]

```
template<typename T , size_t N>
void rti::core::fill_array (
            dds::core::array< T, N > & array,
            const T & value )
```

Initializes all the elements of an array to a value

Referenced by **fill_array()**.

7.21.3.2 fill_array() [2/2]

```
template<typename FinalType , typename ArrayType , size_t N, size_t M>
void rti::core::fill_array (
            dds::core::array< dds::core::array< ArrayType, N >, M > & multi_array,
            const FinalType & value )
```

Initializes all the elements of an array of arbitrary dimensions

References **fill_array()**.

7.21.3.3 initialize_native_array()

```
template<typename CType , typename T , size_t N, typename InitFunc >
void rti::core::initialize_native_array (
            dds::core::array< T, N > & array,
            InitFunc init_func )
```

Initializes all the elements of an array using a C function

7.21.3.4 operator<<() [1/2]

```
template<typename T , size_t N>
std::ostream & rti::core::operator<< (
            std::ostream & out,
            const  bounded_sequence< T, N > & v )
```

Print a vector applying << to all of its elements.

7.21.3.5 bind_listener() [1/2]

```
template<typename Entity , typename  Listener >
ListenerBinder< Entity,  Listener > bind_listener (
            Entity entity,
             Listener * the_listener,
             dds::core::status::StatusMask mask )
```

Sets the listener and creates a **ListenerBinder** (p. 1641) that automatically resets it when all references go out of scope.

[DEPRECATED] The use of **ListenerBinder** (p. 1641) is no longer required as long as the a `std::shared_ptr` is used to set an Entity's listener.

The **ListenerBinder** (p. 1641) created from this function does not delete the listener, it simply sets the listener back to NULL in the Entity. The application is responsible for the life-cycle of the listener.

To also delete the listener when all references go out of scope, see **bind_and_manage_listener()** (p. 709).

```
MyListener my_listener;
dds::sub::DataReader<Foo> reader(subscriber, topic);
// reader will have no listener after scoped_listener (and any other
// references created from scoped_listener) go out of scope--note that
// my_listener is a stack variable so scoped_listener should not delete it
auto scoped_listener = bind_listener(
    reader, &my_listener, dds::core::status::StatusMask::data_available());
```

7.21.3.6 bind_listener() [2/2]

```
template<typename Entity , typename  Listener >
ListenerBinder< Entity,  Listener > bind_listener (
             Entity entity,
              Listener * the_listener )
```

Sets the listener and creates a **ListenerBinder** (p. 1641) that automatically resets it when all references go out of scope.

[DEPRECATED] The use of **ListenerBinder** (p. 1641) is no longer required as long as the a `std::shared_ptr` is used to set an Entity's listener.

This overload works for listeners that don't use a mask

See also

bind_listener(Entity, Listener*, dds::core::status::StatusMask) (p. 708)

7.21.3.7 bind_and_manage_listener() [1/2]

```
template<typename Entity , typename  Listener >
ListenerBinder< Entity,  Listener > bind_and_manage_listener (
             Entity entity,
              Listener * the_listener,
              dds::core::status::StatusMask mask )
```

Sets the listener and creates a **ListenerBinder** (p. 1641) that automatically resets and deletes it when all references go out of scope.

[DEPRECATED] The use of **ListenerBinder** (p. 1641) is no longer required as long as the a `std::shared_ptr` is used to set an Entity's listener.

Example:

```
dds::sub::DataReader<Foo> reader(subscriber, topic);
// The instance of MyListener we are creating and attaching to reader will be
// unset and deleted when scoped_listener (and any other references create
from
// scoped_listener) go out of scope
auto scoped_listener = bind_and_manage_listener(
    reader, new MyListener(),
dds::core::status::StatusMask::data_available());
```

See also

bind_listener() (p. 708)

7.21.3.8 bind_and_manage_listener() [2/2]

```
template<typename Entity , typename  Listener >
ListenerBinder< Entity,  Listener > bind_and_manage_listener (
             Entity entity,
              Listener * the_listener )
```

Sets the listener and creates a **ListenerBinder** (p. 1641) that automatically resets and deletes it when all references go out of scope.

[DEPRECATED] The use of **ListenerBinder** (p. 1641) is no longer required as long as the a `std::shared_ptr` is used to set an Entity's listener.

This overload works for listeners that don't use a mask

See also

bind_and_manage_listener(Entity, Listener*, dds::core::status::StatusMask) (p. 709)

7.21.3.9 operator<<() [2/2]

```
template<typename T >
std::ostream & rti::core::operator<< (
             std::ostream & out,
             const  pointer< T > & ptr )
```

Write pointer contents or the string "NULL"

References **dds::core::null**.

7.21.3.10 default_qos_provider_params() [1/2]

```
 QosProviderParams rti::core::default_qos_provider_params ( )
```

<<***extension***>> *(p. 354)* Get the **rti::core::QosProviderParams** (p. 2049) for the default QosProvider

Note

This is a standalone function in the namespace **rti::core** (p. 701)

Returns

rti::core::QosProviderParams (p. 2049)

7.21.3.11 default_qos_provider_params() [2/2]

```
void rti::core::default_qos_provider_params (
            const  rti::core::QosProviderParams & params )
```

<<***extension***>> *(*p. *354)* Set the **rti::core::QosProviderParams** (p. 2049) for the default QosProvider

Note

This is a standalone function in the namespace **rti::core** (p. 701)

Use this function to configure the profiles that **`dds::core::QosProvider::Default()`** (p. 2036) loads.

```
rti::core::QosProviderParams params;
params.url_profile({"my_profiles.xml"});
rti::core::default_qos_provider_params(params);
auto participant_qos =
    dds::core::QosProvider::Default().participant_qos("my_library::my_profile");
```

Parameters

params	The **QosProviderParams** (p. 2049) to set

7.22 rti::core::builtin_profiles Namespace Reference

<<***extension***>> *(*p. *354)* Contains the names of the built-in profiles

Namespaces

- namespace **qos_lib**

  <<***extension***>> *(*p. *354) Contains the names of the built-in profiles in the regular (non-experimental) qos library*
- namespace **qos_lib_exp**

  <<***extension***>> *(*p. *354)* <<***experimental***>> *(*p. *355) Contains the names of the built-in profiles in the experimental qos library.*
- namespace **qos_snippet_lib**

  <<***extension***>> *(*p. *354) Contains the names of the built-in QoS Snippets*

7.22.1 Detailed Description

<<***extension***>> *(*p. *354)* Contains the names of the built-in profiles

See also

BuiltinQosProfilesGroupDocs

7.23 rti::core::builtin_profiles::qos_lib Namespace Reference

<<***extension***>> *(*p. *354)* Contains the names of the built-in profiles in the regular (non-experimental) qos library

Functions

- std::string **library_name** ()

  A library of non-experimental QoS profiles.
- std::string **baseline** ()

  The most up-to-date QoS default values.
- std::string **baseline_5_0_0** ()

  The QoS default values for version 5.0.0.
- std::string **baseline_5_1_0** ()

  The QoS default values for version 5.1.0.
- std::string **baseline_5_2_0** ()

  The QoS default values for version 5.2.0.
- std::string **baseline_5_3_0** ()

  The QoS default values for version 5.3.0.
- std::string **baseline_6_0_0** ()

  The QoS default values for version 6.0.0.
- std::string **baseline_6_1_0** ()

  The QoS default values for version 6.1.0.
- std::string **baseline_7_0_0** ()

  The QoS default values for version 7.0.0.
- std::string **baseline_7_1_0** ()

  The QoS default values for version 7.1.0.
- std::string **generic_common** ()

  A common Participant base profile.
- std::string **generic_monitoring_common** ()

  Enables RTI Monitoring Library.
- std::string **generic_connext_micro_compatibility** ()

  Sets the values necessary to communicate with RTI Connext Micro.
- std::string **generic_connext_micro_compatibility_2_4_9** ()

  Sets the values necessary to communicate with RTI Connext Micro versions 2.4.4 through at least 2.4.9.
- std::string **generic_connext_micro_compatibility_2_4_3** ()

  Sets the values necessary to communicate with RTI Connext Micro versions 2.4.3 and ealier.
- std::string **generic_other_dds_vendor_compatibility** ()

  Sets the values necessary to interoperate with other DDS vendors.
- std::string **generic_510_transport_compatibility** ()

  Sets the values necessary to interoperate with RTI Connext 5.1.0 using the UDPv6 and/or SHMEM transports.
- std::string **generic_security** ()

  Loads the DDS Secure builtin plugins.
- std::string **generic_strict_reliable** ()

  Enables strict reliability.
- std::string **generic_keep_last_reliable** ()

  Enables keep-last reliability.

- std::string **generic_best_effort** ()

  Enables best-effort reliability kind.
- std::string **generic_strict_reliable_high_throughput** ()

  A profile that can be used to achieve high throughput.
- std::string **generic_strict_reliable_low_latency** ()

  A profile that can be used to achieve low latency.
- std::string **generic_participant_large_data** ()

  A common Participant base profile to facilitate sending large data.
- std::string **generic_participant_large_data_monitoring** ()

  Configures Participants for large data and monitoring.
- std::string **generic_strict_reliable_large_data** ()

  Configures endpoints for sending large data with strict reliability.
- std::string **generic_keep_last_reliable_large_data** ()

  Configures endpoints for sending large data with keep-last reliability.
- std::string **generic_strict_reliable_large_data_fast_flow** ()

  Configures strictly reliable communication for large data with a fast flow controller.
- std::string **generic_strict_reliable_large_data_medium_flow** ()

  Configures strictly reliable communication for large data with a medium flow controller.
- std::string **generic_strict_reliable_large_data_slow_flow** ()

  Configures strictly reliable communication for large data with a slow flow controller.
- std::string **generic_keep_last_reliable_large_data_fast_flow** ()

  Configures keep-last reliable communication for large data with a fast flow controller.
- std::string **generic_keep_last_reliable_large_data_medium_flow** ()

  Configures keep-last reliable communication for large data with a medium flow controller.
- std::string **generic_keep_last_reliable_large_data_slow_flow** ()

  Configures keep-last reliable communication for large data with a slow flow controller.
- std::string **generic_keep_last_reliable_transient_local** ()

  Persists the samples of a DataWriter as long as the entity exists.
- std::string **generic_keep_last_reliable_transient** ()

  Persists samples using RTI Persistence Service.
- std::string **generic_keep_last_reliable_persistent** ()

  Persists samples in permanent storage, like a disk, using RTI Persistence Service.
- std::string **generic_auto_tuning** ()

  Enables the Turbo Mode batching and Auto Throttle experimental features.
- std::string **generic_minimal_memory_footprint** ()

  Uses a set of QoS which reduces the memory footprint of the application.
- std::string **generic_monitoring2** ()

  The default QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use.
- std::string **generic_monitoring2_wan** ()

  The QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use to communicate over WAN with a Collector Service.
- std::string **pattern_rpc** ()

  This profile contains the default RPC and Request-Reply QoS settings.
- std::string **pattern_periodic_data** ()

  Used for applications that expect periodic data.
- std::string **pattern_streaming** ()

  Used for applications that stream data.

- std::string **pattern_reliable_streaming** ()

  Used for applications that stream data and require reliable communication.
- std::string **pattern_event** ()

  Used for applications that handle events.
- std::string **pattern_alarm_event** ()

  Used for applications that handle alarm events.
- std::string **pattern_status** ()

  Used for applications whose samples represent statuses.
- std::string **pattern_alarm_status** ()

  Used for applications in which samples represent alarm statuses.
- std::string **pattern_last_value_cache** ()

  Used for applications that only need the last published value.

7.23.1 Detailed Description

<<***extension***>> *(*p. *354)* Contains the names of the built-in profiles in the regular (non-experimental) qos library

See also

BuiltinQosProfilesGroupDocs

7.24 rti::core::builtin_profiles::qos_lib_exp Namespace Reference

<<***extension***>> (p. *354)* <<***experimental***>> *(*p. *355)* Contains the names of the built-in profiles in the experimental qos library.

Functions

- std::string **library_name** ()

  A library of experimental QoS profiles.
- std::string **generic_strict_reliable** ()

  Enables strict reliability.
- std::string **generic_keep_last_reliable** ()

  Enables keep-last reliability.
- std::string **generic_best_effort** ()

  Enables best-effort reliability kind.
- std::string **generic_strict_reliable_high_throughput** ()

  A profile that can be used to achieve high throughput.
- std::string **generic_strict_reliable_low_latency** ()

  A profile that can be used to achieve low latency.
- std::string **generic_participant_large_data** ()

  A common Participant base profile to facilitate sending large data.
- std::string **generic_participant_large_data_monitoring** ()

  Configures Participants for large data and monitoring.

- std::string **generic_strict_reliable_large_data** ()

  Configures endpoints for sending large data with strict reliability.
- std::string **generic_keep_last_reliable_large_data** ()

  Configures endpoints for sending large data with keep-last reliability.
- std::string **generic_strict_reliable_large_data_fast_flow** ()

  Configures strictly reliable communication for large data with a fast flow controller.
- std::string **generic_strict_reliable_large_data_medium_flow** ()

  Configures strictly reliable communication for large data with a medium flow controller.
- std::string **generic_strict_reliable_large_data_slow_flow** ()

  Configures strictly reliable communication for large data with a slow flow controller.
- std::string **generic_keep_last_reliable_large_data_fast_flow** ()

  Configures keep-last reliable communication for large data with a fast flow controller.
- std::string **generic_keep_last_reliable_large_data_medium_flow** ()

  Configures keep-last reliable communication for large data with a medium flow controller.
- std::string **generic_keep_last_reliable_large_data_slow_flow** ()

  Configures keep-last reliable communication for large data with a slow flow controller.
- std::string **generic_keep_last_reliable_transient_local** ()

  Persists the samples of a DataWriter as long as the entity exists.
- std::string **generic_keep_last_reliable_transient** ()

  Persists samples using RTI Persistence Service.
- std::string **generic_keep_last_reliable_persistent** ()

  Persists samples in permanent storage, like a disk, using RTI Persistence Service.
- std::string **generic_auto_tuning** ()

  Enables the Turbo Mode batching and Auto Throttle experimental features.
- std::string **generic_minimal_memory_footprint** ()

  Uses a set of QoS which reduces the memory footprint of the application.
- std::string **pattern_periodic_data** ()

  Used for applications that expect periodic data.
- std::string **pattern_streaming** ()

  Used for applications that stream data.
- std::string **pattern_reliable_streaming** ()

  Used for applications that stream data and require reliable communication.
- std::string **pattern_event** ()

  Used for applications that handle events.
- std::string **pattern_alarm_event** ()

  Used for applications that handle alarm events.
- std::string **pattern_status** ()

  Used for applications whose samples represent statuses.
- std::string **pattern_alarm_status** ()

  Used for applications in which samples represent alarm statuses.
- std::string **pattern_last_value_cache** ()

  Used for applications that only need the last published value.

7.24.1 Detailed Description

<<***extension***>> *(p. 354)* <<***experimental***>> *(p. 355)* Contains the names of the built-in profiles in the experimental qos library.

This qos library is deprecated. These profiles have been moved to **qos_lib** (p. 712).

The library name is **rti::core::builtin_profiles::qos_lib_exp::library_name()** (p. 482)

7.25 rti::core::builtin_profiles::qos_snippet_lib Namespace Reference

<<***extension***>> *(p. 354)* Contains the names of the built-in QoS Snippets

Functions

- std::string **library_name** ()
  A library of QoS Snippets.
- std::string **snippet_optimization_reliability_protocol_common** ()
  QoS Snippet that configures the reliability protocol with a common configuration.
- std::string **snippet_optimization_reliability_protocol_keep_all** ()
  QoS Snippet that configures the reliability protocol for KEEP_ALL.
- std::string **snippet_optimization_reliability_protocol_keep_last** ()
  QoS Snippet that configures the reliability protocol for KEEP_LAST.
- std::string **snippet_optimization_reliability_protocol_high_rate** ()
  QoS Snippet that configures the reliability protocol for sending data at a high rate.
- std::string **snippet_optimization_reliability_protocol_low_latency** ()
  QoS Snippet that configures the reliability protocol for sending data at low latency.
- std::string **snippet_optimization_reliability_protocol_large_data** ()
  QoS Snippet that configures the reliability protocol for large data.
- std::string **snippet_optimization_reliability_protocol_dynamicmemalloc** ()
  Configures the DataWriter and DataReader caches to use dynamic memory allocation for Large Data samples.
- std::string **snippet_optimization_discovery_common** ()
  QoS Snippet that optimizes discovery with a common configuration.
- std::string **snippet_optimization_discovery_participant_compact** ()
  QoS Snippet that optimizes the Participant QoS to send less discovery information.
- std::string **snippet_optimization_discovery_endpoint_fast** ()
  QoS Snippet that optimizes the Endpoint Discovery to be faster.
- std::string **snippet_optimization_transport_large_buffers** ()
  QoS Snippet that increases the Participant default buffer that shmem and udpv4 use.
- std::string **snippet_qos_policy_reliability_reliable** ()
  QoS Snippet that sets RELIABILITY QoS to RELIABLE.
- std::string **snippet_qos_policy_reliability_best_effort** ()
  QoS Snippet that sets RELIABILITY QoS to BEST_EFFORT.
- std::string **snippet_qos_policy_history_keep_last_1** ()
  QoS Snippet that sets HISTORY QoS to KEEP_LAST kind with depth 1.

- std::string **snippet_qos_policy_history_keep_all** ()

  QoS Snippet that sets HISTORY QosPolicy to KEEP_ALL kind.
- std::string **snippet_qos_policy_publish_mode_asynchronous** ()

  QoS Snippet that sets PUBLISH_MODE QosPolicy to ASYNCHRONOUS kind.
- std::string **snippet_qos_policy_durability_transient_local** ()

  QoS Snippet that sets DURABILITY QosPolicy to TRANSIENT_LOCAL kind.
- std::string **snippet_qos_policy_durability_transient** ()

  QoS Snippet that sets DURABILITY QosPolicy to TRANSIENT kind.
- std::string **snippet_qos_policy_durability_persistent** ()

  QoS Snippet that sets DURABILITY QosPolicy to PERSISTENT kind.
- std::string **snippet_qos_policy_batching_enable** ()

  QoS Snippet that sets BATCH QosPolicy to true.
- std::string **snippet_qos_policy_flow_controller_838mbps** ()

  QoS Snippet that configures and set a FlowController of 838 Mbps.
- std::string **snippet_qos_policy_flow_controller_209mbps** ()

  QoS Snippet that configures and sets a FlowController of 209 Mbps.
- std::string **snippet_qos_policy_flow_controller_52mbps** ()

  QoS Snippet that configures and sets a FlowController of 52 Mbps.
- std::string **snippet_feature_auto_tuning_enable** ()

  QoS Snippet that enables auto_throttle and turbo_mode to true.
- std::string **snippet_feature_monitoring_enable** ()

  QoS Snippet that enables the use of the RTI Monitoring Library.
- std::string **snippet_feature_monitoring2_enable** ()

  QoS Snippet that enables the use of the RTI Monitoring Library 2.0.
- std::string **snippet_feature_security_enable** ()

  QoS Snippet that enables security using the Builtin Security Plugins.
- std::string **snippet_feature_topic_query_enable** ()

  QoS Snippet that enables Topic Query.
- std::string **snippet_feature_type_support_v1_enable** ()

  QoS Snippet that enables TypeObject V1 propagation.
- std::string **snippet_feature_type_support_v1_disable** ()

  QoS Snippet that disables TypeObject V1 propagation.
- std::string **snippet_feature_type_support_v2_enable** ()

  QoS Snippet that enables TypeObject V2 propagation.
- std::string **snippet_feature_type_support_v2_disable** ()

  QoS Snippet that disables TypeObject V2 propagation.
- std::string **snippet_transport_tcp_lan_client** ()

  QoS Snippet that configures a TCP LAN Client over DDS.
- std::string **snippet_transport_tcp_wan_symmetric_client** ()

  QoS Snippet that configures a symmetric WAN TCP Client over DDS.
- std::string **snippet_transport_tcp_wan_asymmetric_server** ()

  QoS Snippet that an asymmetric WAN TCP Server over DDS.
- std::string **snippet_transport_tcp_wan_asymmetric_client** ()

  QoS Snippet that configures an asymmetric WAN TCP Client over DDS.
- std::string **snippet_transport_udp_avoid_ip_fragmentation** ()

  QoS Snippet that configures RTI Connext and the UDP built-in transports (UDPv4, UDPv6, UDPv4_WAN) to avoid IP fragmentation.

- std::string **snippet_transport_udp_wan** ()

  QoS Snippet that enables the RTI Real-Time WAN Transport (UDPv4_WAN).
- std::string **snippet_compatibility_connext_micro_version_2_4_3** ()

  QoS Snippet that configures sets LIVELINESS QoS to be compatible with RTI Connext Micro 2.4.3.
- std::string **snippet_compatibility_other_dds_vendors_enable** ()

  QoS Snippet that configures RTI Connext to interoperate with other DDS vendors.
- std::string **snippet_compatibility_5_1_0_transport_enable** ()

  QoS Snippet that configures RTI Connext to interoperate with RTI Connext 5.1.0 and below for UDPv6 and SHMEM transports.

7.25.1 Detailed Description

<<***extension***>> *(*p. *354)* Contains the names of the built-in QoS Snippets

See also

BuiltinQosProfilesGroupDocs

7.26 rti::core::xtypes Namespace Reference

<<***extension***>> *(*p. *354)* Extensions to **dds::core::xtypes** (p. 631)

Classes

- class **DynamicDataInfo**

  Contains information about a DynamicData sample.
- class **DynamicDataMemberInfo**

  Contains information about a DynamicData member.
- class **DynamicDataProperty**

  <<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350) Specifies the properties of a DynamicData object*
- class **DynamicDataTypeSerializationProperty**

  <<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350) Configures aspects of the memory management in the serialization of* ***dds::core::xtypes::DynamicData*** *(*p. *1461) samples.*
- class **DynamicTypePrintFormatProperty**

  <<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350) A collection of attributes used to configure how DynamicTypes will be formatted when converted to strings.*
- class **LoanedDynamicData**

  <<***move-only-type***>> *(*p. *353) Gives temporary access to a member of another DynamicData object.*
- struct **octet_tag_t**

  <<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350) A special template type to obtain an IDL octet DynamicType from primitive_type*

Enumerations

- enum class **DynamicTypePrintKind** {
  idl ,
  xml ,
  json_schema }

  *The different formats to print a **dds::core::xtypes::DynamicType** (p. 1504).*

Functions

- template<typename TopicType >
  DynamicDataImpl **convert** (const TopicType &sample)

  *<<**extension**>> (p. 354) Creates a DynamicData sample from a typed sample*
- const **dds::core::xtypes::DynamicType** & **resolve_alias** (const **dds::core::xtypes::DynamicType** &type)

  *<<**extension**>> (p. 354) If the type is an alias returns its related type recursively until it is not an alias.*
- std::vector< char > & **to_cdr_buffer** (std::vector< char > &buffer, const DynamicData &sample, **dds::core::policy::DataRepresentationId** representation_id= **dds::core::policy::DataRepresentation::auto_id**())

  *<<**extension**>> (p. 354) Serializes a DynamicData sample into CDR format*
- DynamicData & **from_cdr_buffer** (DynamicData &sample, const std::vector< char > &buffer)

  *<<**extension**>> (p. 354) Creates a DynamicData sample by deserializing a CDR buffer'*
- template<typename TopicType >
  TopicType **convert** (const DynamicData &sample)

  *<<**extension**>> (p. 354) Creates a typed sample from a DynamicData sample.*
- template<typename TopicType >
  DynamicData **convert** (const TopicType &sample)

  *<<**extension**>> (p. 354) Creates a DynamicData sample from a typed sample*
- template<typename TopicType >
  bool **can_convert** (const DynamicData &sample)

  *<<**extension**>> (p. 354) Determines if the DynamicType of this sample is equal to the DynamicType of `TopicType`*
- template<typename... Types>
  std::tuple< Types... > **get_tuple** (const **dds::core::xtypes::DynamicData** &data)

  *<<**C++11**>> (p. 353) <<**experimental**>> (p. 355) <<**extension**>> (p. 354) Retrieves all the values of a DynamicData object into an std::tuple at once*
- template<typename... Types>
  void **set_tuple** (**dds::core::xtypes::DynamicData** &data, const std::tuple< Types... > &value)

  *<<**C++11**>> (p. 353) <<**experimental**>> (p. 355) <<**extension**>> (p. 354) Assigns the values of a std::tuple to a DynamicData object at once*
- template<typename... Types>
  dds::core::xtypes::StructType **create_type_from_tuple** (const std::string &name)

  *<<**C++11**>> (p. 353) <<**experimental**>> (p. 355) <<**extension**>> (p. 354) Creates a StructType from a list of types or a std::tuple*
- void **print_idl** (const DynamicType &type, unsigned int indent=0)

  *<<**extension**>> (p. 354) Prints the IDL representation of this DynamicType to the standard output*
- std::ostream & **to_string** (std::ostream &out, const DynamicType &type, const **rti::core::xtypes::DynamicTypePrintFormatProperty** &format= **DynamicTypePrintFormatProperty**())

  *<<**extension**>> (p. 354) Writes the string representation of this DynamicType to an output stream.*
- std::string **to_string** (const DynamicType &type, const **rti::core::xtypes::DynamicTypePrintFormatProperty** &format= **DynamicTypePrintFormatProperty**())

  *<<**extension**>> (p. 354) Creates the string representation of this DynamicType to a string.*

7.26.1 Detailed Description

<<***extension***>> *(p. 354)* Extensions to **dds::core::xtypes** (p. 631)

7.26.2 Enumeration Type Documentation

7.26.2.1 DynamicTypePrintKind

```
enum class  rti::core::xtypes::DynamicTypePrintKind  [strong]
```

The different formats to print a **dds::core::xtypes::DynamicType** (p. 1504).

Enumerator

idl	Indicates that a **dds::core::xtypes::DynamicType** (p. 1504) is to be printed in IDL format.
xml	Indicates that a **dds::core::xtypes::DynamicType** (p. 1504) is to be printed in XML format.
json_schema	Indicates that a **dds::core::xtypes::DynamicType** (p. 1504) is printed in JSON Schema format.

7.26.3 Function Documentation

7.26.3.1 convert() [1/3]

```
template<typename TopicType >
DynamicDataImpl rti::core::xtypes::convert (
            const TopicType & sample )
```

<<***extension***>> *(p. 354)* Creates a DynamicData sample from a typed sample

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

Template Parameters

TopicType	An IDL topic-type

Parameters

sample	The typed sample to convert to DynamicData

Returns

A new DynamicData object whose type is rti::topic::dynamic_type<TopicType> and contains the data from `sample`

7.26.3.2 resolve_alias()

```
const  dds::core::xtypes::DynamicType & rti::core::xtypes::resolve_alias (
            const  dds::core::xtypes::DynamicType & type )
```

<<***extension***>> *(p. 354)* If the type is an alias returns its related type recursively until it is not an alias.

7.26.3.3 to_cdr_buffer()

```
std::vector< char > & rti::core::xtypes::to_cdr_buffer (
            std::vector< char > & buffer,
            const DynamicData & sample,
             dds::core::policy::DataRepresentationId representation_id =  dds::core::policy::↩
DataRepresentation::auto_id() )
```

<<***extension***>> *(p. 354)* Serializes a DynamicData sample into CDR format

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

Parameters

buffer	The output buffer, which will be resized to the correct size
sample	The sample to serialize
representation↩ _id	The data representation used to serialize the data

Returns

A reference to `buffer`

7.26.3.4 from_cdr_buffer()

```
DynamicData & rti::core::xtypes::from_cdr_buffer (
            DynamicData & sample,
            const std::vector< char > & buffer )
```

<<***extension***>> *(p. 354)* Creates a DynamicData sample by deserializing a CDR buffer'

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

Parameters

sample	The sample to deserialize.
buffer	The buffer containing the CDR data

Returns

A reference to `sample`

7.26.3.5 convert() [2/3]

```
template<typename TopicType >
TopicType rti::core::xtypes::convert (
            const DynamicData & sample )
```

<<***extension***>> *(p. 354)* Creates a typed sample from a DynamicData sample.

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

This function is useful when an application uses DynamicData and regular typed Topics at the same time.

For example, given this IDL type:

```
struct Foo {
    long x;
};
```

We can convert from **`Foo`** (p. 1588) and `DynamicData`:

```
using dds::core::xtypes::DynamicData;
// Receive a DynamicData sample
dds::sub::DataReader<DynamicData> reader = // ...;
dds::sub::LoanedSamples<DynamicData> samples = reader.take();
assert (samples.size() > 0);
// Create a Foo sample
const DynamicData& dynamic_data = samples[0].data();
assert (rti::core::xtypes::can_convert<Foo>(dynamic_data));
Foo typed_data = rti::core::xtypes::convert<Foo>(dynamic_data);
assert (typed_data.x() == dynamic_data.value<int32_t>("x"));
```

Precondition

can_convert<TopicType>(sample), otherwise the behavior is undefined.

Template Parameters

TopicType	An IDL topic-type

Parameters

sample	The DynamicData sample to convert to type `TopicType`

Returns

An instance of `TopicType` containing the data in `sample`.

7.26.3.6 convert() [3/3]

```
template<typename TopicType >
DynamicData rti::core::xtypes::convert (
            const TopicType & sample )
```

<<***extension***>> *(p. 354)* Creates a DynamicData sample from a typed sample

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

Template Parameters

TopicType	An IDL topic-type

Parameters

sample	The typed sample to convert to DynamicData

Returns

A new DynamicData object whose type is rti::topic::dynamic_type<TopicType> and contains the data from `sample`

7.26.3.7 can_convert()

```
template<typename TopicType >
bool rti::core::xtypes::can_convert (
            const DynamicData & sample )
```

<<***extension***>> *(p. 354)* Determines if the DynamicType of this sample is equal to the DynamicType of `TopicType`

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

Template Parameters

TopicType	An IDL topic-type

Returns

True if the DynamicType of TopicType is the same as this sample's and therefore it is possible to create an instance of `TopicType` from this DynamicData object.

See also

convert() (p. 1491)

7.26.3.8 get_tuple()

```
template<typename... Types>
std::tuple< Types... > rti::core::xtypes::get_tuple (
            const dds::core::xtypes::DynamicData & data )
```

<<***C++11***>> *(p. 353)* <<***experimental***>> *(p. 355)* <<***extension***>> *(p. 354)* Retrieves all the values of a DynamicData object into an std::tuple at once

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

See also

Manipulating data reflectively using C++ tuples (p. 612)

7.26.3.9 set_tuple()

```
template<typename... Types>
void rti::core::xtypes::set_tuple (
             dds::core::xtypes::DynamicData & data,
            const std::tuple< Types... > & value )
```

*<<**C++11**>> (p. 353) <<**experimental**>> (p. 355) <<**extension**>> (p. 354)* Assigns the values of a std::tuple to a DynamicData object at once

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

See also

Manipulating data reflectively using C++ tuples (p. 612)

7.26.3.10 create_type_from_tuple()

```
template<typename... Types>
dds::core::xtypes::StructType rti::core::xtypes::create_type_from_tuple (
            const std::string & name )
```

*<<**C++11**>> (p. 353) <<**experimental**>> (p. 355) <<**extension**>> (p. 354)* Creates a StructType from a list of types or a std::tuple

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

Template Parameters

Types	A list of types or a std::tuple. The types must be primitive (see **dds::core::xtypes::PrimitiveType** (p. 1959)) or `std::string`.

The StructType members are all default-created (i.e. non-key, non-optional, with default IDs...) and their names are m0, m1, m2, etc.

See also

Create a DynamicType using tuples (p. 610)

7.26.3.11 print_idl()

```
void rti::core::xtypes::print_idl (
            const DynamicType & type,
            unsigned int indent = 0 )
```

<<***extension***>> *(p. 354)* Prints the IDL representation of this DynamicType to the standard output

7.26.3.12 to_string() [1/2]

```
std::ostream & rti::core::xtypes::to_string (
            std::ostream & out,
            const DynamicType & type,
            const  rti::core::xtypes::DynamicTypePrintFormatProperty & format =  DynamicType↩
PrintFormatProperty() )
```

<<***extension***>> *(p. 354)* Writes the string representation of this DynamicType to an output stream.

Parameters

out	The output stream to which the string representation of the DynamicType should be printed.
type	The DynamicType to be printed.
format	The **DynamicTypePrintFormatProperty** (p. 1509) used to define the format of the string.

Returns

A reference to out.

7.26.3.13 to_string() [2/2]

```
std::string rti::core::xtypes::to_string (
            const DynamicType & type,
            const  rti::core::xtypes::DynamicTypePrintFormatProperty & format =  DynamicType↩
PrintFormatProperty() )
```

<<***extension***>> *(p. 354)* Creates the string representation of this DynamicType to a string.

Parameters

type	The DynamicType to be printed.
format	The **DynamicTypePrintFormatProperty** (p. 1509) used to define the format of the string.

Returns

The string representation of the DynamicType.

7.27 rti::domain Namespace Reference

<<***extension***>> *(p. 354)* Extensions to **dds::domain** (p. 634)

Classes

- class **DomainParticipantConfigParams**

  <<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350) Input paramaters for creating a participant from xml configuration. It allows modification of some of the properties of the entities defined in the configuration.*

Functions

- void **banish_ignored_participants** (const **dds::domain::DomainParticipant** &participant)

  <<***extension***>> *(p. 354) Prevents ignored remote DomainParticipants from receiving traffic from the local DomainParticipant.*
- **rti::core::optional_value**< std::string > **discovered_participant_subject_name** (const **dds::domain::DomainParticipant** &participant, const **dds::core::InstanceHandle** &participant_handle)

  <<***extension***>> *(p. 354) Returns **rti::core::policy::EntityName::name** (p. 1531) for the specified DomainParticipant.*
- **dds::core::InstanceHandleSeq discovered_participants_from_subject_name** (const **dds::domain::DomainParticipant** &participant, const **rti::core::optional_value**< std::string > &subject_name)

  <<***extension***>> *(p. 354) Returns a list of discovered DomainParticipant entities that have the given **rti::core::policy::EntityName::name** (p. 1531).*
- **dds::domain::DomainParticipant find_participant_by_name** (const std::string &participant_name)

  <<***extension***>> *(p. 354) Locates an existing **dds::domain::DomainParticipant** (p. 1326) by name.*
- template<typename ParticipantFwdIterator >
  uint32_t **find_participants** (ParticipantFwdIterator begin, uint32_t max_size)

  <<***extension***>> *(p. 354) Retrieves all the participants created by the application up to a maximum number.*
- template<typename ParticipantFwdIterator >
  uint32_t **find_participants** (ParticipantFwdIterator begin)

  <<***extension***>> *(p. 354) Retrieves all the participants created by the application.*
- const **dds::core::xtypes::DynamicType** & **find_type** (const **dds::domain::DomainParticipant** &participant, const std::string &type_name)

  <<***extension***>> *(p. 354) Retrieves a type registered with this participant*
- void **register_type** (**dds::domain::DomainParticipant** &participant, const std::string &name, const **dds::core::xtypes::DynamicType** &type, const **rti::core::xtypes::DynamicDataTypeSerializationProperty** &serialization_property= **rti::core::xtypes::DynamicDataTypeSerializationProperty::DEFAULT**)

  <<***extension***>> *(p. 354) Registers a DynamicType with specific serialization properties*
- template<typename T >
  void **register_type** (const std::string &registered_type_name= **dds::topic::topic_type_name**< T >::value())

  <<***extension***>> *(p. 354) Registers a User-Generated Type with RTI Connext. This function is used along with XML Application Creation.*

7.27.1 Detailed Description

<<***extension***>> *(p. 354)* Extensions to **dds::domain** (p. 634)

7.27.2 Function Documentation

7.27.2.1 banish_ignored_participants()

```
void rti::domain::banish_ignored_participants (
            const  dds::domain::DomainParticipant & participant )
```

<<***extension***>> *(p. 354)* Prevents ignored remote DomainParticipants from receiving traffic from the local DomainParticipant.

```
#include <rti/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **rti::domain** (p. 727)

This method complements **dds::domain::ignore** (p. 635): ignore_participant prevents the local **dds::domain::DomainParticipant** (p. 1326) from processing traffic from the remote DomainParticipant, while this method prevents already ignored remote DomainParticipants from processing traffic from the local DomainParticipant.

Note: this method is currently only supported when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

MT Safety:

Safe.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::PreconditionNotMetError** (p. 1941), **dds::core::NotEnabledError** (p. 1869)

See also

dds::domain::ignore (p. 635)

7.27.2.2 discovered_participant_subject_name()

```
rti::core::optional_value< std::string > rti::domain::discovered_participant_subject_name (
            const  dds::domain::DomainParticipant & participant,
            const  dds::core::InstanceHandle & participant_handle )
```

<<***extension***>> *(p. 354)* Returns **rti::core::policy::EntityName::name** (p. 1531) for the specified DomainParticipant.

```
#include <rti/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **rti::domain** (p. 727)

This operation retrieves the **rti::core::policy::EntityName::name** (p. 1531) of a **dds::domain::DomainParticipant** (p. 1326) that has been discovered on the network. The participant must be in the same domain as the participant on which this operation is invoked and must not have been "ignored" by means of the **dds::domain::ignore** (p. 635) operation.

The `participant_handle` must correspond to such a DomainParticipant. If the `participant_handle` is **dds::core::InstanceHandle::nil()** (p. 1614) or is not a valid **dds::core::InstanceHandle** (p. 1612) for a DomainParticipant, then the operation will fail with **dds::core::InvalidArgumentError** (p. 1619). If the `participant_handle` corresponds to a DomainParticipant that has not been discovered, then the operation will fail with **dds::core::PreconditionNotMetError** (p. 1941).

Use the operation **dds::domain::discovered_participants** (p. 637) to find the **dds::domain::DomainParticipant** (p. 1326) entities that are currently discovered.

Note: this method has different functionality when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

MT Safety:

Safe.

Parameters

participant	The DomainParticipant that has discovered the discovered participant.
participant_handle	<<***in***>> *(p. 355)* **dds::core::InstanceHandle** (p. 1612) of **dds::domain::DomainParticipant** (p. 1326).

Returns

The **rti::core::policy::EntityName::name** (p. 1531) for the `participant_handle`

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::PreconditionNotMetError** (p. 1941) or **dds::core::NotEnabledError** (p. 1869)

See also

rti::core::policy::EntityName::name (p. 1531)

dds::domain::discovered_participants (p. 637)

7.27.2.3 discovered_participants_from_subject_name()

```
dds::core::InstanceHandleSeq rti::domain::discovered_participants_from_subject_name (
            const  dds::domain::DomainParticipant & participant,
            const  rti::core::optional_value< std::string > & subject_name )
```

<<***extension***>> *(p. 354)* Returns a list of discovered DomainParticipant entities that have the given **rti::core::policy::EntityName::name** (p. 1531).

```
#include <rti/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **rti::domain** (p. 727)

This operation retrieves the same list as **dds::domain::discovered_participants** (p. 637), except this list contains only the participants that have the given **rti::core::policy::EntityName::name** (p. 1531).

Note: this method has different functionality when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

MT Safety:

Safe.

Parameters

participant	The DomainParticipant whose discovered participants this operation will look up.
subject_name	<<***in***>> *(p. 355)* The **rti::core::policy::EntityName::name** (p. 1531) by which to filter the list.

Returns

The list of `InstanceHandles` corresponding to participants with the given **rti::core::policy::EntityName::name** (p. 1531).

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869)

7.27.2.4 find_participant_by_name()

```
dds::domain::DomainParticipant rti::domain::find_participant_by_name (
            const std::string & participant_name )
```

<<***extension***>> *(p. 354)* Locates an existing **dds::domain::DomainParticipant** (p. 1326) by name.

Note

`#include <rti/domain/find.hpp>` This function is in the **`rti::domain`** (p. 727) namespace

If no such DomainParticipant exists, the operation will return **dds::core::null** (p. 438).

7.27.2.5 find_participants() [1/2]

```
template<typename ParticipantFwdIterator >
uint32_t rti::domain::find_participants (
            ParticipantFwdIterator begin,
            uint32_t max_size )
```

<<***extension***>> *(p. 354)* Retrieves all the participants created by the application up to a maximum number.

Note

`#include <rti/domain/find.hpp>` This function is in the **`rti::domain`** (p. 727) namespace

If no such DomainParticipant exists, the operation will return **dds::core::null** (p. 438).

DomainParticipants are ordered in the output sequence from the most recently created to the oldest.

MT Safety:

Unsafe. Creating or deleting DomainParticipants at the same time as calling this operation is not safe.

Template Parameters

ParticipantFwdIterator	A forward iterator whose value type is **dds::domain::DomainParticipant** (p. 1326)

Parameters

begin	The iterator where to begin adding DomainParticipants.
max_size	The maximum number of elements to add

Returns

The number of elements added

7.27.2.6 find_participants() [2/2]

```
template<typename ParticipantFwdIterator >
uint32_t rti::domain::find_participants (
            ParticipantFwdIterator begin )
```

<<***extension***>> *(p. 354)* Retrieves all the participants created by the application.

Note

`#include <rti/domain/find.hpp>` This function is in the **`rti::domain`** (p. 727) namespace

If no such DomainParticipant exists, the operation will return **dds::core::null** (p. 438).

DomainParticipants are ordered in the output sequence from the most recently created to the oldest.

MT Safety:

Unsafe. Creating or deleting DomainParticipants at the same time as calling this operation is not safe.

Template Parameters

ParticipantFwdIterator	A forward iterator whose value type is **dds::domain::DomainParticipant** (p. 1326)

Parameters

begin	The iterator where to begin adding DomainParticipants.

Returns

The number of elements added

7.27.2.7 find_type()

```
const  dds::core::xtypes::DynamicType & rti::domain::find_type (
            const  dds::domain::DomainParticipant & participant,
            const std::string & type_name )
```

<<***extension***>> *(p. 354)* Retrieves a type registered with this participant

`#include <rti/domain/find.hpp>`

Note

This is a standalone function in the namespace **rti::domain** (p. 727)

Every data type used in a DomainParticipant has a registered type name, which in most cases is the same as the type's name (the name used to define the type), but it can be different.

Types are registered by the creation of a **dds::topic::Topic** (p. 2474), whose constructors optionally receive a registered type name different from the type's name.

Types can also be registered when parsing an XML configuration and calling **dds::core::QosProvider::create_↩ participant_from_config()** (p. 2046).

Parameters

participant	The DomainParticipant where the type is registered
type_name	The name used to register the type in this participant

Returns

If `type_name` exists in this `participant`, this function returns the DynamicType describing the type. It can be cast down to **dds::core::xtypes::StructType** (p. 2400) or **dds::core::xtypes::UnionType** (p. 2584), depending on whether the type is a struct or a union.

Exceptions

dds::core::Error *(p. 1539)*	If the type doesn't exist or the operation fails for any other reason

Example:

```
dds::domain::DomainParticipant participant(0);
dds::topic::Topic<Foo> topic1(participant, "Foo Topic"); // registered as "Foo"
dds::topic::Topic<Foo> topic2(participant, "MyFoo Topic", "MyFoo"); // registered as "MyFoo"
const auto& type1 = rti::domain::find_type(participant, "Foo");
const auto& type2 = rti::domain::find_type(participant, "MyFoo");
std::cout « type1 « std::endl;
std::cout « type2 « std::endl;
assert(type1 == type2); // Same type registered with different names
// Cast to StructType to access more information about the type
const auto& type1_struct =
        static_cast<const dds::core::xtypes::StructType&>(type1);
std::cout « type1_struct.member(0).name() « std::endl;
```

7.27.2.8 register_type() [1/2]

```
void rti::domain::register_type (
              dds::domain::DomainParticipant & participant,
             const std::string & name,
             const  dds::core::xtypes::DynamicType & type,
             const  rti::core::xtypes::DynamicDataTypeSerializationProperty & serialization_↩
property =  rti::core::xtypes::DynamicDataTypeSerializationProperty::DEFAULT )
```

<<***extension***>> *(p. 354)* Registers a DynamicType with specific serialization properties

Typically you don't need to call this function, since the topic constructor takes care of that automatically. You do need to call this function before creating the topic if you want to change the default data-serialization property.

Calling this function also allows to change the registered name of the type, which by default is `type.name()`.

Parameters

participant	The participant where to register this type
name	The name to use to register this type
type	The type definition
serialization_property	The data-serialization property

See also

dds::core::xtypes::DynamicType (p. 1504)

dds::core::xtypes::DynamicData (p. 1461)

7.27.2.9 register_type() [2/2]

```
template<typename T >
void rti::domain::register_type (
             const std::string & registered_type_name =  dds::topic::topic_type_name<T>::value()
)
```

<<***extension***>> *(p. 354)* Registers a User-Generated Type with RTI Connext. This function is used along with XML Application Creation.

When using XML Application creation, you must use this function to register any user-generated types with RTI Connext before creating your system.

When you don't use XML Application creation, you don't typically need to call this function, since the topic constructor takes care of that automatically.

Template Parameters

T	The user-generated type that is being registered

Parameters

registered_type_name	The name to use when registering the type. This is the name that will be used in your XML configuration file to refer to the type.

See also

dds::domain::DomainParticipant::create_participant_from_config

XML Application Creation (p. 342)

7.28 rti::flat Namespace Reference

<<***extension***>> *(p. 354)* Support for **FlatData Topic-Types** (p. 419)

Classes

- class **AbstractAlignedList**

  Base class of Offsets to sequences and arrays of non-primitive members.
- class **AbstractBuilder**

  *Base class of all **Builders** (p. 409).*
- class **AbstractListBuilder**

  Base class of all array and sequence builders.
- class **AbstractPrimitiveList**

  Base class for Offsets to sequences and arrays of primitive types.
- class **AbstractSequenceBuilder**

  Base class of Builders for sequence members.
- class **AggregationBuilder**

  Base class of struct and union builders.
- class **FinalAlignedArrayOffset**

  Offset to an array of final elements.
- class **FinalArrayOffset**

  Offset to an array of final elements.
- class **FinalOffset**

  The base class of all Offsets to a final struct type.
- class **FinalSequenceBuilder**

  Builds a sequence member of fixed-size elements.
- struct **flat_type_traits**

  *Given a **Sample** (p. 2274), an Offset or a Builder, it allows obtaining the other types.*
- class **MutableArrayBuilder**

  Builds an array member of variable-size elements.
- class **MutableArrayOffset**

  Offset to an array of variable-size elements.
- class **MutableOffset**

  The base class of all Offsets to a final struct type.
- class **MutableSequenceBuilder**

  Builds a sequence member of variable-size elements.
- class **OffsetBase**

  Base class of all Offset types.
- class **PrimitiveArrayOffset**

  Offset to an array of primitive elements.

- struct **PrimitiveConstOffset**

  A const Offset to an optional primitive member.
- struct **PrimitiveOffset**

  An Offset to an optional primitive member.
- class **PrimitiveSequenceBuilder**

  Builds a sequence of primitive members.
- class **PrimitiveSequenceOffset**

  Offset to a sequence of primitive elements.
- class **Sample**

  The generic definition of FlatData topic-types.
- class **SequenceIterator**

  Iterator for collections of Offsets.
- class **SequenceOffset**

  Offset to a sequence of non-primitive elements.
- class **StringBuilder**

  Builds a string.
- class **StringOffset**

  Offset to a string.
- class **UnionBuilder**

  Base class of builders for user-defined mutable unions.

Functions

- template<typename OffsetType >
  flat_type_traits< OffsetType >::plain_type * **plain_cast** (OffsetType &offset)

  Casts into an equivalent plain C++ type.
- template<typename OffsetType >
  const **flat_type_traits**< OffsetType >::plain_type * **plain_cast** (const OffsetType &offset)

  Const version of plain_cast.
- template<typename TopicType >
  rti::flat::flat_type_traits< TopicType >::builder **build_data** (**dds::pub::DataWriter**< TopicType > &writer)

  Begins building a new sample.
- template<typename TopicType >
  void **discard_builder** (**dds::pub::DataWriter**< TopicType > &writer, typename **rti::flat::flat_type_traits**< TopicType >::builder &builder)

  Discards a sample builder.

7.28.1 Detailed Description

<<***extension***>> *(p. 354)* Support for **FlatData Topic-Types** (p. 419)

7.29 rti::pub Namespace Reference

<<***extension***>> *(p. 354)* Extensions to **dds::pub** (p. 645)

Classes

- class **AcknowledgmentInfo**

  <<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) *Information about an application-acknowledged sample*
- class **FlowController**

  <<***extension***>> (p. *354*) <<***reference-type***>> (p. *351*) *A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous* ***dds::pub::DataWriter*** (p. *1138*) *instances are allowed to write data.*
- class **FlowControllerProperty**

  <<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) *Configures a* ***FlowController*** (p. *1572*)
- struct **FlowControllerSchedulingPolicy_def**

  <<***extension***>> (p. *354*) *Kinds of flow controller shceduling policy*
- class **FlowControllerTokenBucketProperty**

  <<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) *Configures a* ***FlowController*** (p. *1572*) *based on a tocken-bucket mechanism*
- class **WriteParams**

  <<***extension***>> (p. *354*) <<***value-type***>> (p. *350*) *Advanced parameters for writing with a DataWriter*

Typedefs

- typedef **dds::core::safe_enum**< **FlowControllerSchedulingPolicy_def** > **FlowControllerSchedulingPolicy**

  <<***extension***>> (p. *354*) *The safe enumeration for* ***FlowControllerSchedulingPolicy_def::type*** (p. *1581*)

Functions

- **dds::pub::Publisher implicit_publisher** (const **dds::domain::DomainParticipant** &dp)

  <<***extension***>> (p. *354*) *Get the implicit* ***dds::pub::Publisher*** (p. *1993*) *for a given* ***dds::domain::DomainParticipant*** (p. *1326*).
- template<typename T >
  dds::topic::ParticipantBuiltinTopicData matched_subscription_participant_data (const **dds::pub::Data↩ Writer**< T > &writer, const **dds::core::InstanceHandle** &handle)

  <<***extension***>> (p. *354*) *This operation retrieves the information on the discovered* ***dds::domain::DomainParticipant*** (p. *1326*) *associated with the subscription that is currently matching with the* ***dds::pub::DataWriter*** (p. *1138*).
- template<typename T >
  rti::core::LocatorSeq **matched_subscriptions_locators** (const **dds::pub::DataWriter**< T > &writer)

  <<***extension***>> (p. *354*) *Retrieve the list of locators for subscriptions currently associated with this DataWriter.*
- template<typename T >
  bool **is_matched_subscription_active** (const **dds::pub::DataWriter**< T > &writer, const **dds::core::↩ InstanceHandle** &handle)

  <<***extension***>> (p. *354*) *Check if a matched subscription is active.*
- template<typename T >
  std::vector< **dds::topic::SubscriptionBuiltinTopicData** > **matched_subscription_data** (const **dds::pub↩ ::DataWriter**< T > &writer)

  <<***extension***>> (p. *354*) *Obtain the SubscriptionBuiltinTopicData for all of the subscriptions matched with a DataWriter.*
- template<typename PublisherForwardIterator >
  uint32_t **find_publishers** (const **dds::domain::DomainParticipant** participant, PublisherForwardIterator begin, uint32_t max_size)

*<<**extension**>> (p. 354) Retrieve all of the **dds::pub::Publisher** (p. 1993) created from this **dds::domain::DomainParticipant** (p. 1326)*

- template<typename PublisherBackInsertIterator >
  uint32_t **find_publishers** (const **dds::domain::DomainParticipant** participant, PublisherBackInsertIterator begin)

  *Retrieve all the **dds::pub::Publisher** (p. 1993) created from this **dds::domain::DomainParticipant** (p. 1326).*
- **dds::pub::Publisher find_publisher** (const **dds::domain::DomainParticipant** participant, const std::string &publisher_name)

  *<<**extension**>> (p. 354) Looks up a **dds::pub::Publisher** (p. 1993) by its entity name within the **dds::domain::DomainParticipant** (p. 1326).*
- template<typename AnyDataWriterBackInsertIterator >
  uint32_t **find_datawriters** (**dds::pub::Publisher** publisher, AnyDataWriterBackInsertIterator begin)

  *Retrieve all the **dds::pub::DataWriter** (p. 1138) created from this **dds::pub::Publisher** (p. 1993).*
- template<typename AnyDataWriterForwardIterator >
  uint32_t **find_datawriters** (**dds::pub::Publisher** publisher, AnyDataWriterForwardIterator begin, uint32_t max_size)

  *Retrieve all the **dds::pub::DataWriter** (p. 1138) created from this **dds::pub::Publisher** (p. 1993).*
- template<typename Writer >
  Writer **find_datawriter_by_topic_name** (**dds::pub::Publisher** publisher, const std::string &topic_name)

  *<<**extension**>> (p. 354) Retrieves a **dds::pub::DataWriter** (p. 1138) with the given name within the **dds::pub::Publisher** (p. 1993)*
- template<typename Writer >
  Writer **find_datawriter_by_name** (**dds::pub::Publisher** publisher, const std::string &datawriter_name)

  *<<**extension**>> (p. 354) Retrieves a **dds::pub::DataWriter** (p. 1138) with the given name within the **dds::pub::Publisher** (p. 1993)*
- template<typename Writer >
  Writer **find_datawriter_by_name** (**dds::domain::DomainParticipant** participant, const std::string &datawriter_name)

  *<<**extension**>> (p. 354) Retrieves a **dds::pub::DataWriter** (p. 1138) within the **dds::domain::DomainParticipant** (p. 1326) with the given name*
- **FlowController find_flow_controller** (**dds::domain::DomainParticipant** participant, const std::string &name)

  *Retrieves an existing **FlowController** (p. 1572).*

7.29.1 Detailed Description

*<<**extension**>> (p. 354)* Extensions to **dds::pub** (p. 645)

7.29.2 Function Documentation

7.29.2.1 implicit_publisher()

```
dds::pub::Publisher rti::pub::implicit_publisher (
            const  dds::domain::DomainParticipant & dp )
```

<<***extension***>> *(p. 354)* Get the implicit **dds::pub::Publisher** (p. 1993) for a given **dds::domain::Domain↩ Participant** (p. 1326).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

If an implicit Publisher does not already exist, this creates one.

The implicit Publisher is created with default **dds::pub::qos::PublisherQos** (p. 2007) and no listener. When a Domain↩ Participant is deleted, if there are no attached **dds::pub::DataWriter** (p. 1138) that belong to the implicit Publisher, the implicit Publisher will be implicitly deleted.

MT Safety:

UNSAFE. It is not safe to create an implicit Publisher while another thread may be simultaneously calling **dds↩ ::domain::DomainParticipant::default_publisher_qos** (p. 1342).

Parameters

dp	The DomainParticipant that the implicit publisher belongs to

Returns

The implicit publisher

7.29.2.2 matched_subscription_participant_data()

```
template<typename T >
dds::topic::ParticipantBuiltinTopicData rti::pub::matched_subscription_participant_data (
            const  dds::pub::DataWriter< T > & writer,
            const  dds::core::InstanceHandle & handle )
```

<<***extension***>> *(p. 354)* This operation retrieves the information on the discovered **dds::domain::Domain↩ Participant** (p. 1326) associated with the subscription that is currently matching with the **dds::pub::DataWriter** (p. 1138).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

The `subscription_handle` must correspond to a subscription currently associated with the **dds::pub::DataWriter** (p. 1138). Otherwise, the operation will fail with **dds::core::InvalidArgumentError** (p. 1619). The operation may also fail with **dds::core::PreconditionNotMetError** (p. 1941) if the subscription corresponds to the same **dds::domain::DomainParticipant** (p. 1326) that the DataWriter belongs to. Use **dds::pub::matched_subscriptions()** (p. 648) to find the subscriptions that are currently matched with the **dds::pub::DataWriter** (p. 1138).

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::NotEnabledError** (p. 1869)

Template Parameters

T	The topic-type that the DataWriter subscribes to

Parameters

writer	The writer to lookup the matched participant data of
handle	The InstanceHandle to a specific subscription. Must correspond to a subscription currently associated with the DataWriter.

Returns

The **dds::topic::ParticipantBuiltinTopicData** (p. 1908) of the DomainParticipant of a matched subscription of a **dds::pub::DataWriter** (p. 1138)

7.29.2.3 matched_subscriptions_locators()

```
template<typename T >
rti::core::LocatorSeq rti::pub::matched_subscriptions_locators (
            const  dds::pub::DataWriter< T > & writer )
```

<<***extension***>> *(p. 354)* Retrieve the list of locators for subscriptions currently associated with this DataWriter.

The locators returned are the ones that are used by the DDS implementation to communicate with the corresponding matched DataReaders.

Template Parameters

T	The topic-type that the DataWriter publishes.

Parameters

writer	The DataWriter.

Returns

The list of locators

7.29.2.4 is_matched_subscription_active()

```
template<typename T >
bool rti::pub::is_matched_subscription_active (
            const  dds::pub::DataWriter< T > & writer,
            const  dds::core::InstanceHandle & handle )
```

<<***extension***>> *(p. 354)* Check if a matched subscription is active.

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Template Parameters

T	The topic-type that the DataWriter publishes.

Parameters

writer	The DataWriter.
handle	The **dds::core::InstanceHandle** (p. 1612) of the matched subscription.

This API is used for querying the endpoint liveliness of a matched subscription. A matched subscription will be marked as inactive when it becomes nonprogessing (e.g., not responding to a DataWriter's heartbeats, or letting its internal queue fill up without taking the available data). Note that if the participant associated with the matched subscription loses liveliness, the **dds::core::InstanceHandle** (p. 1612) will become invalid and this function will fail with **dds::core::InvalidArgumentError** (p. 1619).

Returns

A boolean indicating whether or not the matched subscription is active.

7.29.2.5 matched_subscription_data()

```
template<typename T >
std::vector<  dds::topic::SubscriptionBuiltinTopicData > rti::pub::matched_subscription_data (
            const  dds::pub::DataWriter< T > & writer )
```

<<***extension***>> *(p. 354)* Obtain the SubscriptionBuiltinTopicData for all of the subscriptions matched with a DataWriter.

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

This API retrieves the matched publication data from all of the subscriptions currently matched a DataWriter.

Template Parameters

T	The topic-type that the DataWriter publishes.

Parameters

writer	The DataWriter.

Returns

A std::vector containing all of the matched subscription data.

7.29.2.6 find_publishers() [1/2]

```
template<typename PublisherForwardIterator >
uint32_t rti::pub::find_publishers (
            const  dds::domain::DomainParticipant participant,
            PublisherForwardIterator begin,
            uint32_t max_size )
```

<<***extension***>> *(p. 354)* Retrieve all of the **dds::pub::Publisher** (p. 1993) created from this **dds::domain::Domain↩ Participant** (p. 1326)

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Template Parameters

PublisherForwardIterator	A forward iterator whose value type is **dds::pub::Publisher** (p. 1993)

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Publishers belong to
begin	A forward iterator to the position in the destination container to insert the found Publishers into
max_size	The maximum number of Publishers to add

Returns

The number of found Publishers

See also

Looking up DataWriters (p. 314)

7.29.2.7 find_publishers() [2/2]

```
template<typename PublisherBackInsertIterator >
uint32_t rti::pub::find_publishers (
            const  dds::domain::DomainParticipant participant,
            PublisherBackInsertIterator begin )
```

Retrieve all the **dds::pub::Publisher** (p. 1993) created from this **dds::domain::DomainParticipant** (p. 1326).

Template Parameters

PublisherBackInsertIterator	A back-inserting iterator whose value type is **dds::pub::Publisher** (p. 1993)

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Publishers belong to
begin	A back-inserting iterator to the position in the destination container to insert the found Publishers in

Returns

The number of found Publishers

See also

Looking up DataWriters (p. 314)

7.29.2.8 find_publisher()

```
dds::pub::Publisher rti::pub::find_publisher (
            const  dds::domain::DomainParticipant participant,
            const std::string & publisher_name )
```

<<***extension***>> *(p. 354)* Looks up a **dds::pub::Publisher** (p. 1993) by its entity name within the **dds::domain::DomainParticipant** (p. 1326).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Every **dds::pub::Publisher** (p. 1993) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 532).

This operation retrieves a **dds::pub::Publisher** (p. 1993) within the **dds::domain::DomainParticipant** (p. 1326) given the entity's name. If there are several **dds::pub::Publisher** (p. 1993) with the same name within the **dds::domain::DomainParticipant** (p. 1326), this function returns the first matching occurrence.

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) to look for the **dds::pub::Publisher** (p. 1993) in.
publisher_name	Entity name of the Publisher

Returns

The first Publisher found with the specified name or **dds::core::null** (p. 438) if it is not found.

See also

Looking up DataWriters (p. 314)

7.29.2.9 find_datawriters() [1/2]

```
template<typename AnyDataWriterBackInsertIterator >
uint32_t rti::pub::find_datawriters (
              dds::pub::Publisher publisher,
             AnyDataWriterBackInsertIterator begin )
```

Retrieve all the **dds::pub::DataWriter** (p. 1138) created from this **dds::pub::Publisher** (p. 1993).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Template Parameters

AnyDataWriterBackInsertIterator	An iterator whose `value_type` is **dds::pub::AnyDataWriter** (p. 815)

Parameters

publisher	The **dds::pub::Publisher** (p. 1993) the DataWriters belong to
begin	A back-inserting iterator to the position in the destination container to insert the found DataWriters into

Returns

The number of found DataWriters

See also

Looking up DataWriters (p. 314)

7.29.2.10 find_datawriters() [2/2]

```
template<typename AnyDataWriterForwardIterator >
uint32_t rti::pub::find_datawriters (
              dds::pub::Publisher publisher,
             AnyDataWriterForwardIterator begin,
             uint32_t max_size )
```

Retrieve all the **dds::pub::DataWriter** (p. 1138) created from this **dds::pub::Publisher** (p. 1993).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Template Parameters

AnyDataWriterForwardIterator	An iterator whose `value_type` is **dds::pub::AnyDataWriter** (p. 815)

Parameters

publisher	The **dds::pub::Publisher** (p. 1993) the DataWriters belong to
begin	A forward iterator to the first position in the destination container to copy the found DataWriters into
max_size	The maximum number of DataWriters to return

Returns

The number of found DataWriters

See also

Looking up DataWriters (p. 314)

7.29.2.11 find_datawriter_by_topic_name()

```
template<typename Writer >
Writer rti::pub::find_datawriter_by_topic_name (
              dds::pub::Publisher publisher,
             const std::string & topic_name )
```

<<***extension***>> *(p. 354)* Retrieves a **dds::pub::DataWriter** (p. 1138) with the given name within the **dds::pub::↩ Publisher** (p. 1993)

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Every **dds::pub::DataWriter** (p. 1138) in the system has an entity name which is configured and stored in the <<***extension***>> *(p. 354)* EntityName policy, **ENTITY_NAME** (p. 532).

This operation retrieves the **dds::pub::DataWriter** (p. 1138) within the **dds::pub::Publisher** (p. 1993) whose name matches the one specified. If there are several **dds::pub::DataWriter** (p. 1138) with the same name within the **dds::pub::Publisher** (p. 1993), the operation returns the first matching occurrence.

Template Parameters

Writer	The type of the writer. It can be **dds::pub::AnyDataWriter** (p. 815), or an instantiation of dds::pub::DataWriter<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))

Parameters

publisher	The **dds::pub::Publisher** (p. 1993) that created the DataWriter to find
topic_name	Entity name of the DataWriter to find

Returns

The AnyDataWriter with the given name

See also

Looking up DataWriters (p. 314)

7.29.2.12 find_datawriter_by_name() [1/2]

```
template<typename Writer >
Writer rti::pub::find_datawriter_by_name (
              dds::pub::Publisher publisher,
             const std::string & datawriter_name )
```

<<***extension***>> *(p. 354)* Retrieves a **dds::pub::DataWriter** (p. 1138) with the given name within the **dds::pub::Publisher** (p. 1993)

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Every **dds::pub::DataWriter** (p. 1138) in the system has an entity name which is configured and stored in the <<***extension***>> *(p. 354)* EntityName policy, **ENTITY_NAME** (p. 532).

This operation retrieves the **dds::pub::DataWriter** (p. 1138) within the **dds::pub::Publisher** (p. 1993) whose name matches the one specified. If there are several **dds::pub::DataWriter** (p. 1138) with the same name within the **dds::pub::Publisher** (p. 1993), the operation returns the first matching occurrence.

Template Parameters

Writer	The type of the writer. It can be **dds::pub::AnyDataWriter** (p. 815), or an instantiation of dds::pub::DataWriter<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))

Parameters

publisher	The **dds::pub::Publisher** (p. 1993) that created the DataWriter to find
datawriter_name	Entity name of the DataWriter to find

Returns

The WRITER with the given name

See also

Looking up DataWriters (p. 314)

7.29.2.13 find_datawriter_by_name() [2/2]

```
template<typename Writer >
Writer rti::pub::find_datawriter_by_name (
              dds::domain::DomainParticipant participant,
             const std::string & datawriter_name )
```

<<***extension***>> *(p. 354)* Retrieves a **dds::pub::DataWriter** (p. 1138) within the **dds::domain::DomainParticipant** (p. 1326) with the given name

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Every **dds::pub::DataWriter** (p. 1138) in the system has an entity name which is configured and stored in the Entity↩ Name policy, **ENTITY_NAME** (p. 532).

Every **dds::pub::Publisher** (p. 1993) in the system has an entity name which is also configured and stored in the EntityName policy.

This operation retrieves a **dds::pub::DataWriter** (p. 1138) within a **dds::pub::Publisher** (p. 1993) given the specified name which encodes both to the **dds::pub::DataWriter** (p. 1138) and the **dds::pub::Publisher** (p. 1993) name.

If there are several **dds::pub::DataWriter** (p. 1138) with the same name within the corresponding **dds::pub::Publisher** (p. 1993) this function returns the first matching occurrence.

The specified name might be given as a fully-qualified entity name or as a plain name.

The fully qualified entity name is a concatenation of the **dds::pub::Publisher** (p. 1993) to which the **dds::pub::Data↩ Writer** (p. 1138) belongs and the entity name of of the **dds::pub::DataWriter** (p. 1138) itself, separated by a double colon "::". For example: MyPublisherName::MyDataWriterName

The plain name contains the **dds::pub::DataWriter** (p. 1138) name only. In this situation it is implied that the **dds::pub↩ ::DataWriter** (p. 1138) belongs to the implicit **dds::pub::Publisher** (p. 1993) so the use of a plain name is equivalent to specifying a fully qualified name with the **dds::pub::Publisher** (p. 1993) name part being "implicit". For example: the plain name "MyDataWriterName" is equivalent to specifiying the fully qualified name "implicit::MyDataWriterName"

The **dds::pub::DataWriter** (p. 1138) is only looked up within the **dds::pub::Publisher** (p. 1993) specified in the fully qualified name, or within the implicit **dds::pub::Publisher** (p. 1993) if the name was not fully qualified.

Template Parameters

Writer	The type of the writer. It can be **dds::pub::AnyDataWriter** (p. 815), or an instantiation of dds::pub::DataWriter<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) within which the **dds::pub::DataWriter** (p. 1138) exists
datawriter_name	**EntityName** (p. 1530) of the DataWriter to find

Returns

The WRITER with the given name

See also

implicit_publisher(const dds::domain::DomainParticipant& dp) (p. 738);

Looking up DataWriters (p. 314)

7.29.2.14 find_flow_controller()

```
FlowController find_flow_controller (
            dds::domain::DomainParticipant participant,
           const std::string & name )
```

Retrieves an existing **FlowController** (p. 1572).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Parameters

participant	The DomainParticipant associated to the **FlowController** (p. 1572)
name	The name used to create the **FlowController** (p. 1572) or the name of one of the built-in FlowControllers (**FlowController::DEFAULT_NAME** (p. 58), **FlowController::FIXED_RATE_NAME** (p. 59), **FlowController::ON_DEMAND_NAME** (p. 60))

Returns

The flow controller with that name in that participant or an empty reference (equals to **dds::core::null** (p. 438)) if it doesn't exist

See also

FlowController::retain() (p. 1576)

7.30 rti::sub Namespace Reference

<<***extension***>> *(p. 354)* Extensions to **dds::sub** (p. 658)

Classes

- class **AckResponseData**

  <<***extension***>> *(p. 354) Data payload associated to an application-level acknowledgment*
- struct **IsValidData**

  <<***extension***>> *(p. 354) A functor that returns true when a sample has valid data.*
- class **LoanedSample**

  *The element type of a **dds::sub::LoanedSamples** (p. 1663) collection.*
- class **ManipulatorSelector**
- class **SampleIterator**

  *A random-access iterator of **LoanedSample** (p. 1659).*
- class **SampleProcessor**

  <<***extension***>> *(p. 354)* <<***reference-type***>> *(p. 351) Utility to read and process the data samples that one or more DataReaders receive using a sample handler.*
- class **SharedSamples**

  Provides access to a collection of middleware-loaned samples.
- class **TopicQuery**

  <<***extension***>> *(p. 354)* <<***reference-type***>> *(p. 351) Allows a **dds::sub::DataReader** (p. 976) to query the sample cache of its matching dds::pub::DataWriters.*
- class **TopicQueryData**

  <<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350) Provides information about a **TopicQuery** (p. 2516)*
- class **TopicQuerySelection**

  <<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350) Specifies the data query that defines a **TopicQuery** (p. 2516).*
- struct **TopicQuerySelectionKind_def**

  *The definition of the **dds::core::safe_enum** (p. 2265) **rti::sub::TopicQuerySelectionKind** (p. 71).*
- class **ValidLoanedSamples**

  <<***extension***>> *(p. 354)* <<***C++11***>> *(p. 353)* <<***move-only-type***>> *(p. 353) Provides access to only those samples that contain valid data*
- class **ValidSampleIterator**

  A forward iterator adapter that skips invalid samples.

Typedefs

- typedef **dds::core::safe_enum**< **TopicQuerySelectionKind_def** > **TopicQuerySelectionKind**

  Safe Enumeration *(p. 429) of* ***TopicQuerySelectionKind_def*** *(p. 2528)*

Functions

- **dds::sub::Subscriber implicit_subscriber** (const **dds::domain::DomainParticipant** &dp)

  <<***extension***>> *(p. 354) Retrieves the implicit* ***dds::sub::Subscriber*** *(p. 2410) for the given* ***dds::domain::Domain↩ Participant*** *(p. 1326).*

- template<typename T >
  dds::topic::ParticipantBuiltinTopicData matched_publication_participant_data (const **dds::sub::Data↩ Reader**< T > &reader, const **dds::core::InstanceHandle** &handle)

  <<***extension***>> *(p. 354) This operation retrieves the information on the discovered* ***dds::domain::DomainParticipant*** *(p. 1326) associated with the publication that is currently matching with the* ***dds::sub::DataReader*** *(p. 976).*

- template<typename T >
  bool **is_matched_publication_alive** (const **dds::sub::DataReader**< T > &reader, const **dds::core::↩ InstanceHandle** &handle)

  <<***extension***>> *(p. 354) Check if a matched publication is alive.*

- template<typename T >
  std::vector< **dds::topic::PublicationBuiltinTopicData** > **matched_publication_data** (const **dds::sub::↩ DataReader**< T > &reader)

  <<***extension***>> *(p. 354) Obtain the PublicationBuiltinTopicData for all of the publications matched with a DataReader.*

- template<typename SubscriberForwardIterator >
  uint32_t **find_subscribers** (const **dds::domain::DomainParticipant** participant, SubscriberForwardIterator **begin**, uint32_t max_size)

  <<***extension***>> *(p. 354) Retrieve all of the* ***dds::sub::Subscriber*** *(p. 2410) created from this* ***dds::domain::Domain↩ Participant*** *(p. 1326)*

- template<typename SubscriberBackInsertIterator >
  uint32_t **find_subscribers** (const **dds::domain::DomainParticipant** participant, SubscriberBackInsertIterator **begin**)

  <<***extension***>> *(p. 354) Retrieve all of the* ***dds::sub::Subscriber*** *(p. 2410) created from this* ***dds::domain::Domain↩ Participant*** *(p. 1326)*

- **dds::sub::Subscriber find_subscriber** (const **dds::domain::DomainParticipant** participant, const std::string &subscriber_name)

  <<***extension***>> *(p. 354) Finds a Subscriber by name*

- template<typename AnyDataReaderBackInsertIterator >
  uint32_t **find_datareaders** (**dds::sub::Subscriber** subscriber, AnyDataReaderBackInsertIterator **begin**)

  <<***extension***>> *(p. 354) Retrieve all the* ***dds::sub::DataReader*** *(p. 976) created from this* ***dds::sub::Subscriber*** *(p. 2410)*

- template<typename AnyDataReaderForwardIterator >
  uint32_t **find_datareaders** (**dds::sub::Subscriber** subscriber, AnyDataReaderForwardIterator **begin**, uint32↩ _t max_size)

  <<***extension***>> *(p. 354) Retrieve all the readers created from a subscriber.*

- template<typename Reader >
  Reader **find_datareader_by_topic_name** (**dds::sub::Subscriber** subscriber, const std::string &topic_name)

  <<***extension***>> *(p. 354) Retrieves a* ***dds::sub::DataReader*** *(p. 976) with the given topic name within the* ***dds::sub↩ ::Subscriber*** *(p. 2410)*

- template<typename Reader >
  Reader **find_datareader_by_name** (**dds::sub::Subscriber** subscriber, const std::string &datareader_name)

<<**extension**>> (p. 354) Retrieves a **dds::sub::DataReader** (p. 976) with the given name within the **dds::sub::Subscriber** (p. 2410)

- template<typename Reader , typename T >
  Reader **find_datareader_by_topic_description** (const **dds::sub::Subscriber** &subscriber, const **dds::topic::TopicDescription**< T > &topic_description)

  <<**extension**>> (p. 354) Retrieves a **dds::sub::DataReader** (p. 976) with the given TopicDescription within the **dds::sub::Subscriber** (p. 2410)
- template<typename Reader >
  Reader **find_datareader_by_name** (**dds::domain::DomainParticipant** participant, const std::string &datareader_name)

  <<**extension**>> (p. 354) Retrieves a **dds::sub::DataReader** (p. 976) within the **dds::domain::DomainParticipant** (p. 1326) with the given name
- template<typename T >
  dds::sub::Sample< T > **copy_to_sample** (const **rti::sub::LoanedSample**< T > &ls)

  Copies the contents of a **rti::sub::LoanedSample** (p. 1659) into a **dds::sub::Sample** (p. 2270).
- template<typename T >
  std::ostream & **operator**<< (std::ostream &out, const **LoanedSample**< T > &sample)

  Calls the operator on the data or prints [invalid data].
- template<typename T >
  ValidLoanedSamples< T >::iterator **begin** (**ValidLoanedSamples**< T > &ls)
- template<typename T >
  ValidLoanedSamples< T >::const_iterator **begin** (const **ValidLoanedSamples**< T > &ls)
- template<typename T >
  ValidLoanedSamples< T >::iterator **end** (**ValidLoanedSamples**< T > &ls)
- template<typename T >
  ValidLoanedSamples< T >::const_iterator **end** (const **ValidLoanedSamples**< T > &ls)
- template<typename T >
  void **swap** (**ValidLoanedSamples**< T > &ls1, **ValidLoanedSamples**< T > &ls2) throw ()
- template<typename T >
  ValidLoanedSamples< T > **valid_data** (LoanedSamples< T > &&samples)

  <<**C++11**>> (p. 353) <<**extension**>> (p. 354) Returns a collection that provides access only to samples with valid data
- bool **operator==** (const SampleInfoImpl &other) const

  Compare two **dds::sub::SampleInfo** (p. 2285) objects for equality.
- template<typename T >
  ValidSampleIterator< T > **valid_data** (const **SampleIterator**< T > &sample_iterator)

  <<**extension**>> (p. 354) Returns an iterator that skips invalid samples
- **TopicQueryData** **create_topic_query_data_from_service_request** (const **rti::topic::ServiceRequest** &service_request)

  Creates a **TopicQueryData** (p. 2520) from a ServiceRequest.
- **TopicQuery** **find_topic_query** (**dds::sub::AnyDataReader** datareader, const **rti::core::Guid** &guid)

  Looks up a **TopicQuery** (p. 2516) by its GUID.
- template<typename T >
  void **unpack** (const **dds::sub::SharedSamples**< T > &samples, std::vector< std::shared_ptr< const T > > &sample_vector)

  <<**extension**>> (p. 354) <<**C++11**>> (p. 353) Unpacks a **SharedSamples** (p. 2367) collection into individual shared_ptr's in a vector
- template<typename T >
  std::vector< std::shared_ptr< const T > > **unpack** (const **dds::sub::SharedSamples**< T > &samples)

  <<**extension**>> (p. 354) <<**C++11**>> (p. 353) Unpacks a **SharedSamples** (p. 2367) collection into individual shared_ptr's in a vector

- template<typename T >
  std::vector< std::shared_ptr< const T > > **unpack** (**dds::sub::LoanedSamples**< T > &&samples)

  <<***extension***>> *(*p. *354)* <<***C++11***>> *(*p. *353) Unpacks a LoanedSamples collection into individual shared_ptr's in a vector*

7.30.1 Detailed Description

<<***extension***>> *(*p. *354)* Extensions to **dds::sub** (p. 658)

7.30.2 Function Documentation

7.30.2.1 implicit_subscriber()

```
dds::sub::Subscriber rti::sub::implicit_subscriber (
            const  dds::domain::DomainParticipant & dp )
```

<<***extension***>> *(*p. *354)* Retrieves the implicit **dds::sub::Subscriber** (p. 2410) for the given **dds::domain::↩ DomainParticipant** (p. 1326).

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

If an implicit Subscriber does not already exist, this creates one.

The implicit Subscriber is created with default **dds::sub::qos::SubscriberQos** (p. 2423) and no listener. When a DomainParticipant is deleted, if there are no attached **dds::sub::DataReader** (p. 976) that belong to the implicit Subscriber, the implicit Subscriber will be implicitly deleted.

MT Safety:

UNSAFE. it is not safe to create the implicit subscriber while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_subscriber_qos(const dds::sub::qos::SubscriberQos & qos)** (p. 1343).

Parameters

dp	The DomainParticipant that the implicit subscriber belongs to.

Returns

The implicit Subscriber

7.30.2.2 matched_publication_participant_data()

```
template<typename T >
dds::topic::ParticipantBuiltinTopicData rti::sub::matched_publication_participant_data (
            const  dds::sub::DataReader< T > & reader,
            const  dds::core::InstanceHandle & handle )
```

<<***extension***>> *(p. 354)* This operation retrieves the information on the discovered **dds::domain::Domain↩ Participant** (p. 1326) associated with the publication that is currently matching with the **dds::sub::DataReader** (p. 976).

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

The `publication_handle` must correspond to a publication currently associated with the **dds::sub::DataReader** (p. 976). Otherwise, the operation will fail with **dds::core::InvalidArgumentError** (p. 1619). The operation may also fail with **dds::core::PreconditionNotMetError** (p. 1941) if the publication corresponds to the same **dds::domain::↩ DomainParticipant** (p. 1326) that the DataReader belongs to. Use the operation **dds::sub::matched_publications** (p. 668) to find the publications that are currently matched with the **dds::sub::DataReader** (p. 976).

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869)

Template Parameters

T	The topic-type that the DataReader subscribes to

Parameters

reader	The reader to lookup the matched participant data of
handle	The InstanceHandle to a specific publication. Must correspond to a publication currently associated with the DataReader. This handle is available in the **dds::sub::SampleInfo::publication_handle()** (p. 2290)

Returns

The **dds::topic::ParticipantBuiltinTopicData** (p. 1908) of the DomainParticipant of a matched publication of a **dds::sub::DataReader** (p. 976)

7.30.2.3 is_matched_publication_alive()

```
template<typename T >
bool rti::sub::is_matched_publication_alive (
            const  dds::sub::DataReader< T > & reader,
            const  dds::core::InstanceHandle & handle )
```

<<***extension***>> *(p. 354)* Check if a matched publication is alive.

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

T	The topic-type that the DataReader subscribes to.

Parameters

reader	The DataReader.
handle	The **dds::core::InstanceHandle** (p. 1612) of the matched publication.

This API is used for querying the endpoint liveliness of a matched publication. A matched publication will be marked as not alive if the liveliness that it committed to through its **LIVELINESS** (p. 535) QoS policy was not respected. Note that if the participant associated with the matched publication loses liveliness, the **dds::core::InstanceHandle** (p. 1612) will become invalid and this function will fail with **dds::core::InvalidArgumentError** (p. 1619).

of the matched publication. See **dds::sub::matched_publications** (p. 668) for a description of what is considered a matched publication.

Exceptions

One	of the **Standard Exceptions** (p. 428)

Returns

A boolean indicating whether or not the matched publication is alive.

7.30.2.4 matched_publication_data()

```
template<typename T >
std::vector<  dds::topic::PublicationBuiltinTopicData > rti::sub::matched_publication_data (
            const  dds::sub::DataReader< T > & reader )
```

<<***extension***>> *(p. 354)* Obtain the PublicationBuiltinTopicData for all of the publications matched with a DataReader.

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

This API retrieves the matched subscription data from all of the publications currently matched a DataReader.

Template Parameters

T	The topic-type that the DataReader subscribes to.

Parameters

reader	The DataReader.

Returns

A std::vector containing all of the matched publication data.

7.30.2.5 find_subscribers() [1/2]

```
template<typename SubscriberForwardIterator >
uint32_t rti::sub::find_subscribers (
            const  dds::domain::DomainParticipant participant,
            SubscriberForwardIterator begin,
            uint32_t max_size )
```

<<***extension***>> *(p. 354)* Retrieve all of the **dds::sub::Subscriber** (p. 2410) created from this **dds::domain::DomainParticipant** (p. 1326)

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

SubscriberForwardIterator	Type of the forward iterator passed into this function, whose value_type must be **dds::sub::Subscriber** (p. 2410).

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Publishers belong to
begin	A forward iterator to the position in the destination container where to begin copying the found Subscribers into
max_size	The maximum size of Subscribers to add

Returns

The number of found Subscribers

7.30.2.6 find_subscribers() [2/2]

```
template<typename SubscriberBackInsertIterator >
uint32_t rti::sub::find_subscribers (
            const  dds::domain::DomainParticipant participant,
            SubscriberBackInsertIterator begin )
```

<<***extension***>> *(p. 354)* Retrieve all of the **dds::sub::Subscriber** (p. 2410) created from this **dds::domain::DomainParticipant** (p. 1326)

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

AnyDataReaderBackInsertIterator	Type of the back-inserting iterator passed into this function, whose value_type must be **dds::sub::Subscriber** (p. 2410).

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Subscribers belong to
begin	A back-inserting iterator to the position in the destination container to insert the found Subscribers into

Returns

The number of found Subscribers

7.30.2.7 find_subscriber()

```
dds::sub::Subscriber rti::sub::find_subscriber (
            const  dds::domain::DomainParticipant participant,
            const std::string & subscriber_name )
```

<<***extension***>> *(p. 354)* Finds a Subscriber by name

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

AnyDataReaderBackInsertIterator	Type of the back-inserting iterator passed into this function, whose value_type must be **dds::sub::AnyDataReader** (p. 807).

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Subscribers belong to
subscriber_name	The name of the Subscriber

Returns

A valid reference if the a subscriber with that name exists or a reference equals to **dds::core::null** (p. 438) otherwise.

7.30.2.8 find_datareaders() [1/2]

```
template<typename AnyDataReaderBackInsertIterator >
uint32_t rti::sub::find_datareaders (
             dds::sub::Subscriber subscriber,
            AnyDataReaderBackInsertIterator begin )
```

<<***extension***>> *(p. 354)* Retrieve all the **dds::sub::DataReader** (p. 976) created from this **dds::sub::Subscriber** (p. 2410)

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

AnyDataReaderBackInsertIterator	Type of the back-inserting iterator passed into this function, whose value_type must be **dds::sub::AnyDataReader** (p. 807).

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) the DataReaders belong to
begin	A back-inserting iterator to the position in the destination container to insert the found DataReaders into

Returns

The number of found DataReaders

See also

Looking up DataReaders (p. 322)

7.30.2.9 find_datareaders() [2/2]

```
template<typename AnyDataReaderForwardIterator >
uint32_t rti::sub::find_datareaders (
             dds::sub::Subscriber subscriber,
            AnyDataReaderForwardIterator begin,
            uint32_t max_size )
```

<<***extension***>> *(*p. *354)* Retrieve all the readers created from a subscriber.

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

AnyDataReaderForwardIterator	A forward iterator whose `value_type` is **dds::sub::AnyDataReader** (p. 807)

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) the readers belong to
begin	A forward iterator to the position in the destination container where to begin copying the found readers into.
max_size	The maximum number of readers to return

Returns

The number of found readers

See also

Looking up DataReaders (p. 322)

7.30.2.10 find_datareader_by_topic_name()

```
template<typename Reader >
Reader rti::sub::find_datareader_by_topic_name (
```

```
    dds::sub::Subscriber subscriber,
const std::string & topic_name )
```

<<***extension***>> *(p. 354)* Retrieves a **dds::sub::DataReader** (p. 976) with the given topic name within the **dds::sub::Subscriber** (p. 2410)

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Use this operation on the built-in **dds::sub::Subscriber** (p. 2410) (**Built-in Topics** (p. 46)) to access the built-in **dds::sub::DataReader** (p. 976) entities for the built-in topics.

The built-in **dds::sub::DataReader** (p. 976) is created when this operation is called on a built-in topic for the first time. The built-in **dds::sub::DataReader** (p. 976) is deleted automatically when the **dds::domain::DomainParticipant** (p. 1326) is deleted.

To ensure that builtin **dds::sub::DataReader** (p. 976) entities receive all the discovery traffic, it is suggested that you lookup the builtin **dds::sub::DataReader** (p. 976) before the **dds::domain::DomainParticipant** (p. 1326) is enabled. Looking up builtin **dds::sub::DataReader** (p. 976) may implicitly register builtin transports due to creation of **dds::sub::DataReader** (p. 976) (see **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered). Therefore, if you are want to modify builtin transport properties, do so *before* using this operation.

Therefore the suggested sequence when looking up builtin DataReaders is:

- Create a disabled **dds::domain::DomainParticipant** (p. 1326).
- (optional) Modify builtin transport properties
- Call **dds::sub::builtin_subscriber()** (p. 671).
- Call **dds::sub::find()** (p. 672).
- Call **enable()** (p. 581) on the DomainParticipant.

The returned **dds::sub::DataReader** (p. 976) may be enabled or disabled.

If more than one **dds::sub::DataReader** (p. 976) is attached to the **dds::sub::Subscriber** (p. 2410), this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **dds::sub::DataReader** (p. 976) in one thread while another thread is simultaneously creating or destroying that **dds::sub::DataReader** (p. 976).

Template Parameters

Reader	The type of the reader. It can be **dds::sub::AnyDataReader** (p. 807), or an instantiation of dds::sub::DataReader<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) that created the DataReader to find
topic_name	Topic name of the DataReader to find

Returns

The DataReader with the given topic name, or **dds::core::null** (p. 438) if it doesn't exist

See also

Looking up DataReaders (p. 322)

7.30.2.11 find_datareader_by_name() [1/2]

```
template<typename Reader >
Reader rti::sub::find_datareader_by_name (
              dds::sub::Subscriber subscriber,
             const std::string & datareader_name )
```

<<***extension***>> *(p. 354)* Retrieves a **dds::sub::DataReader** (p. 976) with the given name within the **dds::sub::Subscriber** (p. 2410)

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

Reader	The type of the reader. It can be **dds::sub::AnyDataReader** (p. 807), or an instantiation of dds::sub::DataReader<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))

Every **dds::sub::DataReader** (p. 976) in the system has an entity name which is configured and stored in the <<***extension***>> *(p. 354)* EntityName policy, **ENTITY_NAME** (p. 532).

This operation retrieves the **dds::sub::DataReader** (p. 976) within the **dds::sub::Subscriber** (p. 2410) whose name matches the one specified. If there are several **dds::sub::DataReader** (p. 976) with the same name within the **dds::sub::Subscriber** (p. 2410), the operation returns the first matching occurrence.

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) that created the DataReader to find
datareader_name	Entity name of the DataReader to find

Returns

The DataReader with the given name, or **dds::core::null** (p. 438) if it doesn't exist

See also

Looking up DataReaders (p. 322)

7.30.2.12 find_datareader_by_topic_description()

```
template<typename Reader , typename T >
Reader rti::sub::find_datareader_by_topic_description (
            const  dds::sub::Subscriber & subscriber,
            const  dds::topic::TopicDescription< T > & topic_description )
```

<<***extension***>> *(p. 354)* Retrieves a **dds::sub::DataReader** (p. 976) with the given TopicDescription within the **dds::sub::Subscriber** (p. 2410)

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

Reader	The type of the reader returned, for example, dds::sub::DataReader<Foo>, or **dds::sub::AnyDataReader** (p. 807)
T	The topic-type

Parameters

subscriber	The Subscriber to which the DataReader belongs
topic_description	The TopicDescription identifying the DataReader to find

Returns

The found DataReader, or **dds::core::null** (p. 438) if it doesn't exist

7.30.2.13 find_datareader_by_name() [2/2]

```
template<typename Reader >
Reader rti::sub::find_datareader_by_name (
```

```
dds::domain::DomainParticipant participant,
const std::string & datareader_name )
```

<<***extension***>> *(p. 354)* Retrieves a **dds::sub::DataReader** (p. 976) within the **dds::domain::DomainParticipant** (p. 1326) with the given name

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

Reader	The type of the reader. It can be **dds::sub::AnyDataReader** (p. 807), or an instantiation of dds::sub::DataReader<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))

Every **dds::sub::DataReader** (p. 976) in the system has an entity name which is configured and stored in the Entity↩ Name policy, **ENTITY_NAME** (p. 532).

Every **dds::sub::Subscriber** (p. 2410) in the system has an entity name which is also configured and stored in the EntityName policy, **ENTITY_NAME** (p. 532).

This operation retrieves a **dds::sub::DataReader** (p. 976) within a **dds::sub::Subscriber** (p. 2410) given the specified name which encodes both to the **dds::sub::DataReader** (p. 976) and the **dds::sub::Subscriber** (p. 2410) name.

If there are several **dds::sub::DataReader** (p. 976) with the same name within the corresponding **dds::sub::Subscriber** (p. 2410) this function returns the first matching occurrence.

The specified name might be given as a fully-qualified entity name or as a plain name.

The fully qualified entity name is a concatenation of the **dds::sub::Subscriber** (p. 2410) to which the **dds::sub::Data↩ Reader** (p. 976) belongs and the entity name of of the **dds::sub::DataReader** (p. 976) itself, separated by a double colon "::". For example: MySubscriberName::MyDataReaderName

The plain name contains the **dds::sub::DataReader** (p. 976) name only. In this situation it is implied that the **dds::sub↩ ::DataReader** (p. 976) belongs to the implicit **dds::sub::Subscriber** (p. 2410) so the use of a plain name is equivalent to specifying a fully qualified name with the **dds::sub::Subscriber** (p. 2410) name part being "implicit". For example: the plain name "MyDataReaderName" is equivalent to specifiying the fully qualified name "implicit::MyDataReaderName"

The **dds::sub::DataReader** (p. 976) is only looked up within the **dds::sub::Subscriber** (p. 2410) specified in the fully qualified name, or within the implicit **dds::sub::Subscriber** (p. 2410) if the name was not fully qualified.

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) within which the **dds::sub::DataReader** (p. 976) exists
datareader_name	EntityName of the DataReader to find

Returns

The first reader with the given name or **dds::core::null** (p. 438) if it is not found.

See also

rti::sub::find_datareader_by_name(const dds::sub::Subscriber&, const std::string&)

7.30.2.14 copy_to_sample()

```
template<typename T >
dds::sub::Sample< T > copy_to_sample (
            const  rti::sub::LoanedSample< T > & ls )
```

Copies the contents of a **rti::sub::LoanedSample** (p. 1659) into a **dds::sub::Sample** (p. 2270).

Example:

```
dds::sub::LoanedSamples<Foo> samples = reader.take();
std::vector<Sample<Foo> > sample_vector;
std::transform(
    rti::sub::valid_data(samples.begin()),
    rti::sub::valid_data(samples.end()),
    std::back_inserter(sample_vector),
    rti::sub::copy_to_sample<Foo>);
```

7.30.2.15 operator<<()

```
template<typename T >
std::ostream & operator<< (
            std::ostream & out,
            const  LoanedSample< T > & sample )
```

Calls the operator on the data or prints [invalid data].

7.30.2.16 begin() [1/2]

```
template<typename T >
ValidLoanedSamples< T >::iterator begin (
             ValidLoanedSamples< T > & ls )
```

See also

ValidLoanedSamples::begin() (p. 2597)

Referenced by **dds::sub::DataReader< T >::find_datareaders()**, and **dds::sub::Subscriber::find_subscribers()**.

7.30.2.17 begin() [2/2]

```
template<typename T >
ValidLoanedSamples< T >::const_iterator begin (
            const  ValidLoanedSamples< T > & ls )
```

See also

ValidLoanedSamples::begin() (p. 2597)

7.30.2.18 end() [1/2]

```
template<typename T >
ValidLoanedSamples< T >::iterator end (
             ValidLoanedSamples< T > & ls )
```

See also

ValidLoanedSamples::end() (p. 2598)

7.30.2.19 end() [2/2]

```
template<typename T >
ValidLoanedSamples< T >::const_iterator end (
            const  ValidLoanedSamples< T > & ls )
```

See also

ValidLoanedSamples::end() (p. 2598)

7.30.2.20 swap()

```
template<typename T >
void swap (
             ValidLoanedSamples< T > & ls1,
             ValidLoanedSamples< T > & ls2 ) throw ( )
```

See also

ValidLoanedSamples::swap() (p. 2600)

7.30.2.21 valid_data() [1/2]

```
template<typename T >
ValidLoanedSamples< T > rti::sub::valid_data (
            LoanedSamples< T > && samples )
```

*<<**C++11**>> (p. 353) <<**extension**>> (p. 354)* Returns a collection that provides access only to samples with valid data

Template Parameters

T	The topic-type. It has to match the type of the DataReader.

This function transforms a LoanedSamples collection into another collection whose iterators only access valid-data samples, skipping any sample such that !sample.info().valid().

This operation is O(1) and will not copy the data samples or allocated any additional memory.

The typical way to use this function is to directly call it on the return value of a **read()** (p. 1024)/take() operation and use it in a for-loop. For example:

```
auto valid_samples = rti::sub::valid_data(reader.read());
for (auto sample : valid_samples) {
    // no need to check sample.info().valid()
    std::cout « sample.data() « std::endl;
}
```

Parameters

samples	The collection of LoanedSamples to transform into a **ValidLoanedSamples** (p. 2595). It must be an rvalue, so valid actual parameters are the result of one of the read/take operations:
	`auto vs = rti::sub::valid_data(reader.take());`
	Or an `std::move'd` existing collection:
	`auto ls = reader.take(); auto vs = rti::sub::valid_data(std::move(ls));`
	`// 'ls' is now invalid and can't be further used`

Returns

A forward-iterable collection that provides access only to samples with valid data. Note that this collection doesn't provide random access.

Postcondition

`samples` is invalid cannot be used after this call

See also

rti::sub::valid_data(const SampleIterator<T>&) (p. 767), which applies to an iterator rather to the whole collection

Reading data samples (p. 318)

7.30.2.22 operator==()

```
bool rti::sub::operator== (
            const SampleInfoImpl & other ) const
```

Compare two **dds::sub::SampleInfo** (p. 2285) objects for equality.

7.30.2.23 valid_data() [2/2]

```
template<typename T >
ValidSampleIterator< T > rti::sub::valid_data (
            const  SampleIterator< T > & sample_iterator )
```

<<***extension***>> *(p. 354)* Returns an iterator that skips invalid samples

Given a regular sample iterator, this functions creates another iterator `it` that behaves exactly the same except that `it++` moves to the next valid sample (or to the end of the collection). That is, if `it` doesn't point to the end of the collection, `it->info.valid()` is always true.

This is useful when your application doesn't need to deal with samples containing meta-information only.

For example, the following code copies all the data in a LoanedSamples collection skipping any invalid samples (otherwise, attempting to copy the data from an invalid sample would throw an exception, see rti::sub::LoanedSample::operator const DataType& ()).

```
dds::sub::LoanedSamples<KeyedType> samples = reader.take();
std::vector<KeyedType> data_vector;
std::copy(
     rti::sub::valid_data(samples.begin()),
     rti::sub::valid_data(samples.end()),
     std::back_inserter(data_vector));
```

Note that `valid_data`(samples.begin()) won't point to the first element if that element is not a valid sample.

A similar utility is the functor **rti::sub::IsValidData** (p. 1624).

See also

dds::sub::LoanedSamples (p. 1663)

rti::sub::IsValidData (p. 1624)

dds::sub::SampleInfo::valid() (p. 2289)

rti::sub::valid_data(LoanedSamples<T>&&) (p. 766), which applies to the whole collection instead of an iterator

Reading data samples (p. 318)

7.30.2.24 create_topic_query_data_from_service_request()

```
TopicQueryData create_topic_query_data_from_service_request (
            const  rti::topic::ServiceRequest & service_request )
```

Creates a **TopicQueryData** (p. 2520) from a ServiceRequest.

This operation will extract the content from the request body of the **rti::topic::ServiceRequest** (p. 2358) to create a **rti::sub::TopicQueryData** (p. 2520) object.

The specified **rti::topic::ServiceRequest** (p. 2358) must be a valid sample associated with the service id **rti::core::ServiceRequestId_def::TOPIC_QUERY** (p. 2362). Otherwise this operation will return false.

This operation can be called within the context of a **dds::pub::DataWriterListener::on_service_request_accepted** (p. 1209) to retrieve the **rti::sub::TopicQueryData** (p. 2520) of a **rti::topic::ServiceRequest** (p. 2358) that has been received with service id **rti::core::ServiceRequestId_def::TOPIC_QUERY** (p. 2362)

Parameters

service_request	The **rti::topic::ServiceRequest** (p. 2358) that contains the **rti::sub::TopicQueryData** (p. 2520) as part of its request body.

Returns

A **rti::sub::TopicQueryData** (p. 2520) object where the content from the service request is extracted.

7.30.2.25 find_topic_query()

```
TopicQuery find_topic_query (
             dds::sub::AnyDataReader datareader,
            const  rti::core::Guid & guid )
```

Looks up a **TopicQuery** (p. 2516) by its GUID.

Parameters

datareader	The DataReader used to create the **TopicQuery** (p. 2516)
guid	The **TopicQuery** (p. 2516)'s GUID

Returns

The **TopicQuery** (p. 2516) if it exists or a **dds::core::null** (p. 438) reference otherwise.

See also

rti::sub::TopicQuery::guid() (p. 2518)

7.30.2.26 unpack() [1/3]

```
template<typename T >
void rti::sub::unpack (
            const  dds::sub::SharedSamples< T > & samples,
            std::vector< std::shared_ptr< const T > > & sample_vector )
```

<<***extension***>> *(p. 354)* <<***C++11***>> *(p. 353)* Unpacks a **SharedSamples** (p. 2367) collection into individual shared_ptr's in a vector

Note

`#include <rti/sub/unpack.hpp>`

This is a standalone function in the namespace **rti::sub** (p. 751)

This function creates a reference (not a copy) to each sample with valid data in a **SharedSamples** (p. 2367) container and pushes it back into a vector.

Each individual sample in the vector retains a reference to the original **SharedSamples** (p. 2367) that controls when the loan is returned. These references can be further shared. When all the references go out of scope, the loan is returned.

This can be also useful to insert samples from different calls to **read()** (p. 1024)/take() into the same vector. It is however recommended to not hold these samples indefinitely, since they use internal resources.

Example:

```
dds::sub::SharedSamples<Foo> shared_samples = reader.take();
std::vector<std::shared_ptr<const Foo» sample_vector;
rti::sub::unpack(shared_samples, sample_vector);
std::cout « *sample_vector[0] « std::endl;
// ...
// Read more samples, unpack them at the end of the same vector
shared_samples = reader.take();
rti::sub::unpack(shared_samples, sample_vector);
// References to the samples can be shared freely
std::shared_ptr<const Foo> sample = sample_vector[3];
// ...
// The loans will be returned automatically
```

Note

To finalize a DataReader, all the shared_ptr obtained via **unpack()** (p. 769) need to have been released. Otherwise DataReader::close() will fail with **dds::core::PreconditionNotMetError** (p. 1941).

Template Parameters

T	The topic-type

Parameters

samples	The collection of samples obtained from the DataReader
sample_vector	The destination where the samples are pushed back.

Referenced by **dds::sub::SharedSamples< T, DELEGATE >::unpack()**.

7.30.2.27 unpack() [2/3]

```
template<typename T >
std::vector< std::shared_ptr< const T > > rti::sub::unpack (
            const  dds::sub::SharedSamples< T > & samples )
```

*<<**extension**>> (p. 354) <<**C++11**>> (p. 353)* Unpacks a **SharedSamples** (p. 2367) collection into individual shared_ptr's in a vector

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

This overload returns a new vector instead of adding into an existing one.

See also

unpack(const dds::sub::SharedSamples<T>&, std::vector<std::shared_ptr<const T> >&) (p. 769)

7.30.2.28 unpack() [3/3]

```
template<typename T >
std::vector< std::shared_ptr< const T > > rti::sub::unpack (
            dds::sub::LoanedSamples< T > && samples )
```

<<***extension***>> *(p. 354)* <<***C++11***>> *(p. 353)* Unpacks a LoanedSamples collection into individual shared_ptr's in a vector

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

This overload is a shortcut for `unpack(SharedSamples<T>(loaned_samples))` to simplify code like the following:

```
auto sample_vector = unpack(reader.take());
```

See also

unpack(const dds::sub::SharedSamples<T>&, std::vector<std::shared_ptr<const T> >&) (p. 769)

7.31 rti::topic Namespace Reference

<<***extension***>> *(p. 354)* Extensions to **dds::topic** (p. 688)

Classes

- class **ContentFilter**

  *<<**extension**>> (p. 354) A class to inherit from when implementing a custom content filter*
- class **CustomFilter**

  *<<**extension**>> (p. 354) <<**reference-type**>> (p. 351) A wrapper class for the user-defined implementation of a **ContentFilter** (p. 950).*
- struct **dynamic_type**

  Provides a DynamicType that represents an IDL-generated type.
- class **ExpressionProperty**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Provides additional information about the filter expression passed to the writer_compile method of **rti::topic::WriterContentFilter** (p. 2652)*
- struct **extensibility**

  *<<**extension**>> (p. 354) Indicates the extensibility kind of a topic-type*
- class **FilterSampleInfo**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) Provides meta information associated with the sample.*
- struct **no_compile_data_t**

  *The type to specify as the CompileData template parameter to your **ContentFilter** (p. 950) if your compile function does not return any data.*
- struct **PrintFormatKind_def**

  *The definition of the **dds::core::safe_enum** (p. 2265) PrintFormatKind.*
- class **PrintFormatProperty**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A collection of attributes used to configure how data samples will be formatted when converted to a string.*
- class **ServiceRequest**

  *<<**extension**>> (p. 354) <<**value-type**>> (p. 350) A request coming from one of the built-in services*
- struct **topic_type_disabled_copy**

  *<<**extension**>> (p. 354) Indicates whether a TopicType is uncopyable*
- struct **topic_type_has_external_members**

  *<<**extension**>> (p. 354) Indicates if a topic type contains directly or indirectly IDL external members.*
- class **WriterContentFilter**

  *<<**extension**>> (p. 354) A class to inherit from when implementing a writer-side custom content filter*
- class **WriterContentFilterHelper**

  *<<**extension**>> (p. 354) A class to inherit from when implementing a writer-side custom content filter.*

Typedefs

- typedef **dds::core::safe_enum**< **PrintFormatKind_def** > **PrintFormatKind**

  ***Safe Enumeration** (p. 429) of **PrintFormatKind_def** (p. 1961) Format kinds available when converting data samples to string representations.*

Functions

- template<typename TopicType >
  void **from_cdr_buffer_no_alloc** (TopicType &sample, const std::vector< char > &buffer)

  Deserializes a sample from a buffer of bytes in CDR format.
- template<typename TopicType >
  TopicType **from_cdr_buffer** (const std::vector< char > &buffer)

  Creates a sample by deserializing a buffer of bytes in CDR format.
- template<typename TopicType >
  std::vector< char > & **to_cdr_buffer** (std::vector< char > &buffer, const TopicType &sample, **dds::core::policy::DataRepresentationId** representation)

  Serializes a sample into a buffer of octets in CDR format.
- template<typename TopicType >
  std::vector< char > & **to_cdr_buffer** (std::vector< char > &buffer, const TopicType &sample)

  Serializes a sample into a buffer of octets in CDR format.
- std::string **sql_filter_name** ()

  <<***extension***>> (p. *354) The name of the built-in SQL filter*
- std::string **stringmatch_filter_name** ()

  <<***extension***>> (p. *354) The name of the built-in StringMatch filter*
- template<typename AnyTopicBackInsertIterator >
  uint32_t **find_topics** (**dds::domain::DomainParticipant** participant, AnyTopicBackInsertIterator begin)

  <<***extension***>> (p. *354) Retrieve all the **dds::topic::Topic** (p. 2474) created from this **dds::domain::DomainParticipant** (p. 1326)*
- template<typename AnyTopicForwardIterator >
  uint32_t **find_topics** (**dds::domain::DomainParticipant** participant, AnyTopicForwardIterator begin, uint32_t max_size)

  <<***extension***>> (p. *354) Retrieve all the **dds::topic::Topic** (p. 2474) created from this **dds::domain::DomainParticipant** (p. 1326)*
- template<typename T >
  CustomFilter< T > **find_content_filter** (const **dds::domain::DomainParticipant** &participant, const std::string &filter_name)

  *Lookup a content filter previously registered with **dds::domain::DomainParticipant::register_contentfilter** (p. 1349).*
- template<typename FwdIterator >
  uint32_t **find_registered_content_filters** (const **dds::domain::DomainParticipant** &participant, FwdIterator begin, uint32_t max_size)

  <<***extension***>> (p. *354) Lookup the names of up to max_size number of the custom content filters registered to a **dds::domain::DomainParticipant** (p. 1326)*
- template<typename BinIterator >
  uint32_t **find_registered_content_filters** (const **dds::domain::DomainParticipant** &participant, BinIterator begin)

  *Lookup the names of all of the custom content filters registered to a **dds::domain::DomainParticipant** (p. 1326).*
- template<typename TopicType >
  std::ostream & **to_string** (std::ostream &out, const TopicType &sample, const **PrintFormatProperty** &print_format= **PrintFormatProperty::Default**())

  Prints a data sample to an output stream.
- template<typename TopicType >
  std::string **to_string** (const TopicType &sample, const **PrintFormatProperty** &print_format= **PrintFormatProperty::Default**())

  Tranforms a data sample into a human-readable string format.
- template<typename TopicType >
  TopicType **from_string** (RTI_STRING_IN str, **PrintFormatKind** format_kind=PrintFormatKind::JSON)

Creates a data sample from a JSON string representation.

- std::string **service_request_topic_name** ()

  ServiceRequest *(p. 2358) built-in topic name.*

7.31.1 Detailed Description

<<***extension***>> *(p. 354)* Extensions to **dds::topic** (p. 688)

7.31.2 Function Documentation

7.31.2.1 find_topics() [1/2]

```
template<typename AnyTopicBackInsertIterator >
uint32_t rti::topic::find_topics (
             dds::domain::DomainParticipant participant,
            AnyTopicBackInsertIterator begin )
```

<<***extension***>> *(p. 354)* Retrieve all the **dds::topic::Topic** (p. 2474) created from this **dds::domain::Domain↩ Participant** (p. 1326)

```
#include <dds/topic/find.hpp>
```

Note

This is a standalone function in the namespace **rti::topic** (p. 771)

Template Parameters

AnyTopicBackInsertIterator	Type of the back-inserting iterator, whose value_type is **dds::topic::AnyTopic** (p. 825).

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Topics belong to
begin	A back-inserting iterator to the position in the destination container to insert the found Topics into

Returns

The number of found Topics

7.31.2.2 find_topics() [2/2]

```
template<typename AnyTopicForwardIterator >
uint32_t rti::topic::find_topics (
             dds::domain::DomainParticipant participant,
            AnyTopicForwardIterator begin,
            uint32_t max_size )
```

<<***extension***>> *(p. 354)* Retrieve all the **dds::topic::Topic** (p. 2474) created from this **dds::domain::Domain↩ Participant** (p. 1326)

```
#include <dds/topic/find.hpp>
```

Note

This is a standalone function in the namespace **rti::topic** (p. 771)

Template Parameters

AnyTopicBackInsertIterator	Type of the back-inserting iterator, whose value_type is **dds::topic::AnyTopic** (p. 825).

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Topics belong to
begin	A forward iterator to the position in the destination container to insert the found Topics in
max_size	The maximum number of Topics to return

Returns

The number of found Topics

7.31.2.3 find_registered_content_filters() [1/2]

```
template<typename FwdIterator >
uint32_t rti::topic::find_registered_content_filters (
            const  dds::domain::DomainParticipant & participant,
            FwdIterator begin,
            uint32_t max_size )
```

<<***extension***>> *(p. 354)* Lookup the names of up to max_size number of the custom content filters registered to a **dds::domain::DomainParticipant** (p. 1326)

Note

This is a standalone function in the namespace **rti::topic** (p. 771)

The names of the RTI Connext built-in content filters will not be returned as part of the list.

Template Parameters

FwdIterator	A forwward iterator whose value type is `std::string` (or convertible to)

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) that the content filters are registered with
begin	A forward iterator to the position in the destination container to insert the names of the found content filters into
max_size	The maximum number of filters to lookup

Returns

The number of found content filters

Referenced by **dds::topic::ContentFilteredTopic< T >::find_registered_content_filters()**.

7.31.2.4 find_registered_content_filters() [2/2]

```
template<typename BinIterator >
uint32_t rti::topic::find_registered_content_filters (
             const  dds::domain::DomainParticipant & participant,
             BinIterator begin )
```

Lookup the names of all of the custom content filters registered to a **dds::domain::DomainParticipant** (p. 1326).

```
#include <dds/topic/find.hpp>
```

Note

This is a standalone function in the namespace **rti::topic** (p. 771)

The names of the RTI Connext built-in content filters will not be returned as part of the list.

Template Parameters

BinIterator	A back-inserting iterator whose value type is `std::string` (or convertible to)

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) that the content filters are registered with
begin	A back-inserting iterator to the position in the destination container to insert the the names of the found content filters into

Returns

The number of found content filters

7.32 rti::util Namespace Reference

<<***extension***>> *(*p. *354)* Contains general-purpose utilities

Namespaces

- namespace **activation**

  <<***extension***>> *(*p. *354) Contains the necessary functions related to Activation.*
- namespace **discovery**

  <<***extension***>> *(*p. *354) Contains the functions to take discovery snapshot for DomainParticipant, DataWriter and DataReader entities.*
- namespace **function_history**

  <<***extension***>> *(*p. *354) Contains the functions to enable Function History feature.*
- namespace **heap_monitoring**

  <<***extension***>> *(*p. *354) Contains the functions to enable and use the heap-monitoring utility.*
- namespace **network_capture**

  <<***extension***>> *(*p. *354) Contains the functions to enable and use the Network Capture utility.*

Classes

- class **atomic_shared_ptr**

  Utility class to provide atomic operations on std::shared_ptr.
- class **StreamFlagSaver**

Functions

- void **sleep** (const **dds::core::Duration** &durationIn)

  Blocks the calling thread for the specified duration.
- uint64_t **spin_per_microsecond** ()

  Returns the number of spin operations needed to wait 1 microsecond.
- void **spin** (uint64_t spin_count)

  Performs a spin operation (active wait) as many times as indicated.

7.32.1 Detailed Description

<<***extension***>> *(*p. *354)* Contains general-purpose utilities

7.33 rti::util::activation Namespace Reference

<<***extension***>> *(*p. *354)* Contains the necessary functions related to Activation.

Functions

- bool **unload_activation_library** ()

  Unloads the Activation Library (rti_activation) if it was previously loaded.

7.33.1 Detailed Description

<<***extension***>> *(*p. *354)* Contains the necessary functions related to Activation.

7.34 rti::util::discovery Namespace Reference

<<***extension***>> *(*p. *354)* Contains the functions to take discovery snapshot for DomainParticipant, DataWriter and DataReader entities.

Functions

- void **take_snapshot** (**dds::domain::DomainParticipant** participant)

  <<***extension***>> *(*p. *354) Take a snapshot of the remote participants discovered by a local one.*
- void **take_snapshot** (**dds::domain::DomainParticipant** participant, const std::string &file_name)

  <<***extension***>> *(*p. *354) Take a snapshot of the remote participants discovered by a local one.*
- void **take_snapshot** (**dds::pub::AnyDataWriter** writer)

  <<***extension***>> *(*p. *354) Take a snapshot of the compatible and incompatible remote readers matched by a local writer.*
- void **take_snapshot** (**dds::pub::AnyDataWriter** writer, const std::string &file_name)

  <<***extension***>> *(*p. *354) Take a snapshot of the compatible and incompatible remote readers matched by a local writer.*
- void **take_snapshot** (**dds::sub::AnyDataReader** reader)

  <<***extension***>> *(*p. *354) Take a snapshot of the compatible and incompatible remote writers matched by a local reader.*
- void **take_snapshot** (**dds::sub::AnyDataReader** reader, const std::string &file_name)

  <<***extension***>> *(*p. *354) Take a snapshot of the compatible and incompatible remote writers matched by a local reader.*

7.34.1 Detailed Description

<<***extension***>> *(*p. *354)* Contains the functions to take discovery snapshot for DomainParticipant, DataWriter and DataReader entities.

7.35 rti::util::function_history Namespace Reference

<<***extension***>> *(*p. *354)* Contains the functions to enable Function History feature.

7.35.1 Detailed Description

<<***extension***>> *(*p. *354)* Contains the functions to enable Function History feature.

7.36 rti::util::heap_monitoring Namespace Reference

<<***extension***>> *(*p. *354)* Contains the functions to enable and use the heap-monitoring utility.

Classes

- class **HeapMonitoringParams**

  Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.
- struct **SnapshotContentFormat_def**

  Bitmap used to decide which information of the snapshot will be displayed.
- struct **SnapshotOutputFormat_def**

  Specify the format of the output of the snapshot. RTI Connext.

Typedefs

- typedef **dds::core::safe_enum**< **SnapshotOutputFormat_def** > **SnapshotOutputFormat**

  Specify the format of the output of the snapshot.
- typedef **dds::core::safe_enum**< **SnapshotContentFormat_def** > **SnapshotContentFormat**

  Bitmap used to decide which information of the snapshot will be displayed.

Functions

- bool **enable** ()

  Starts monitoring the heap memory used by RTI Connext.
- bool **enable** (const **HeapMonitoringParams** &params)

  Starts monitoring the heap memory used by RTI Connext with params.
- void **disable** ()

  Stops monitoring the heap memory used by RTI Connext.
- bool **pause** ()

  Pauses heap monitoring.
- bool **resume** ()

  Resumes heap monitoring.
- bool **take_snapshot** (const std::string &filename, bool print_details=false)

  Saves the current heap memory usage in a file.

7.36.1 Detailed Description

<<***extension***>> *(*p. *354)* Contains the functions to enable and use the heap-monitoring utility.

7.37 rti::util::network_capture Namespace Reference

<<***extension***>> *(*p. *354)* Contains the functions to enable and use the Network Capture utility.

Classes

- class **ContentKindMask**

  <<***extension***>> *(*p. *354) Mask indicating the types of contents to remove from RTPS frames before saving them to the capture file.*
- class **NetworkCaptureParams**

  <<***extension***>> *(*p. *354) Input parameters for starting Network Capture.*
- class **TrafficKindMask**

  <<***extension***>> *(*p. *354) Mask indicating the traffic direction to capture.*

Functions

- bool **enable** ()

  Enable Network Capture.
- bool **disable** ()

  Disable Network Capture.
- bool **set_default_params** (const **NetworkCaptureParams** &params)

  Set the default Network Capture parameters.
- bool **start** (const std::string &filename)

  Start Network Capture.
- bool **start** (**dds::domain::DomainParticipant** participant, const std::string &filename)

  Start Network Capture for a participant.
- bool **start** (const std::string &filename, const **NetworkCaptureParams** &params)

  Start Network Capture with parameters.
- bool **start** (**dds::domain::DomainParticipant** participant, const std::string &filename, const **NetworkCaptureParams** &params)

  Start Network Capture with parameters for a participant.
- bool **stop** ()

  Stop Network Capture.
- bool **stop** (**dds::domain::DomainParticipant** participant)

  Stop Network Capture.
- bool **pause** ()

  Pause Network Capture.
- bool **pause** (**dds::domain::DomainParticipant** participant)

  Pause Network Capture.
- bool **resume** ()

  Resume Network Capture.
- bool **resume** (**dds::domain::DomainParticipant** participant)

  Resume Network Capture.

7.37.1 Detailed Description

<<***extension***>> *(*p. *354)* Contains the functions to enable and use the Network Capture utility.

Chapter 8

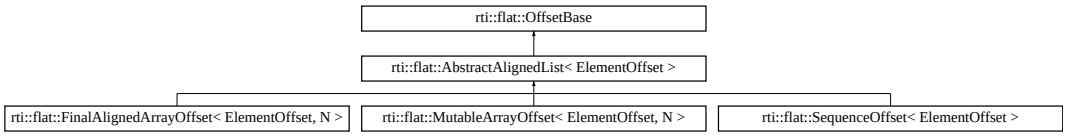
Class Documentation

8.1 rti::flat::AbstractAlignedList< ElementOffset > Class Template Reference

Base class of Offsets to sequences and arrays of non-primitive members.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::AbstractAlignedList< ElementOffset >:



Public Types

- typedef **SequenceIterator**< ElementOffset, typename ElementOffset::offset_kind > **iterator**
  *The iterator type, **SequenceIterator** (p. 2328).*

Public Member Functions

- **iterator begin** ()
  Gets an iterator to the first Offset.
- **iterator end** ()
  Gets an iterator to the past-the-end element.

8.1.1 Detailed Description

template<typename ElementOffset>
class rti::flat::AbstractAlignedList< ElementOffset >

Base class of Offsets to sequences and arrays of non-primitive members.

Template Parameters

ElementOffset	The Offset type of the elements

8.1.2 Member Typedef Documentation

8.1.2.1 iterator

```
template<typename ElementOffset >
typedef SequenceIterator<ElementOffset, typename ElementOffset::offset_kind> rti::flat::↩
AbstractAlignedList< ElementOffset >::iterator
```

The iterator type, **SequenceIterator** (p. 2328).

8.1.3 Member Function Documentation

8.1.3.1 begin()

```
template<typename ElementOffset >
iterator rti::flat::AbstractAlignedList< ElementOffset >::begin ( ) [inline]
```

Gets an iterator to the first Offset.

begin() (p. 782) and **end()** (p. 782) enable the use of range-for loops, for example:

```
SequenceOffset<MyFlatMutableOffset> sequence_offset = my_type_offset.my_sequence();
for (auto element : sequence_offset) {
    std::cout « element.x() « std::endl;
}
```

References **rti::flat::OffsetBase::get_buffer_size()**.

8.1.3.2 end()

```
template<typename ElementOffset >
iterator rti::flat::AbstractAlignedList< ElementOffset >::end ( ) [inline]
```

Gets an iterator to the past-the-end element.

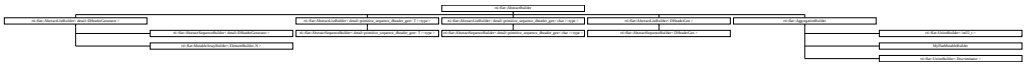
References **rti::flat::OffsetBase::get_buffer_size()**.

8.2 rti::flat::AbstractBuilder Class Reference

Base class of all **Builders** (p. 409).

`#include <Builder.hpp>`

Inheritance diagram for rti::flat::AbstractBuilder:



Public Member Functions

- void **discard** ()

  Discards a member in process of being built.
- bool **is_nested** () const

  Returns whether this is a member Builder.
- bool **is_valid** () const

  Whether this Builder is valid.
- rti::xcdr::length_t **capacity** () const

  Returns the total capacity in bytes.

Protected Member Functions

- virtual ~**AbstractBuilder** ()

  If this is a member Builder, it calls finish().

8.2.1 Detailed Description

Base class of all **Builders** (p. 409).

8.2.2 Constructor & Destructor Documentation

8.2.2.1 ~AbstractBuilder()

`virtual rti::flat::AbstractBuilder::~AbstractBuilder ( )  [inline], [protected], [virtual]`

If this is a member Builder, it calls finish().

If this Builder is building a member (that is, **is_nested()** (p. 784) is true), and the object goes out of scope before finish() has been called, its destructor calls finish(). Note, however, that it won't report any error.

If this Builder is building a sample (!is_nested()), its destructor doesn't do anything.

8.2.3 Member Function Documentation

8.2.3.1 discard()

```
void rti::flat::AbstractBuilder::discard ( )  [inline]
```

Discards a member in process of being built.

This function ends the creation of a member, returning the Builder of the type that contains the member to its previous state, as if this member had never been built.

Precondition

This object must be a member Builder, not a sample Builder.

This method is useful when during the building of a member an error occurs and the application wants to roll back, instead of finishing an incomplete member.

8.2.3.2 is_nested()

```
bool rti::flat::AbstractBuilder::is_nested ( ) const  [inline]
```

Returns whether this is a member Builder.

A member Builder is a Builder that has been created by calling a "`build_<member>`" function on another Builder (for example, **MyFlatMutableBuilder::build_my_mutable()** (p. 1769)).

Returns

True if this is a member Builder, or false if this is a sample Builder.

8.2.3.3 is_valid()

```
bool rti::flat::AbstractBuilder::is_valid ( ) const  [inline]
```

Whether this Builder is valid.

A Builder is not valid when it is default-constructed, or after any of these functions is called: finish(), finish_sample(), **discard()** (p. 784).

8.2.3.4 capacity()

```
rti::xcdr::length_t rti::flat::AbstractBuilder::capacity ( ) const  [inline]
```

Returns the total capacity in bytes.

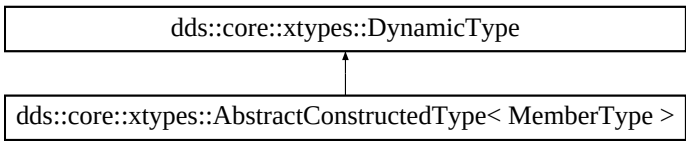
The capacity is the total number of bytes this Builder can contain. For a sample Builder (that is, one such that **is_nested()** (p. 784) returns false), **rti::flat::build_data** (p. 411) reserves enough bytes to accommodate any sample of a given type.

8.3 dds::core::xtypes::AbstractConstructedType< MemberType > Class Template Reference

The base class of types that have members and an extensibility kind.

```
#include <DynamicTypeImpl.hpp>
```

Inheritance diagram for dds::core::xtypes::AbstractConstructedType< MemberType >:



Public Types

- typedef MemberType **Member**

  The member type (specified by the subclasses of this class)
- typedef uint32_t **MemberIndex**

  The member index type.

Public Member Functions

- **dds::core::xtypes::ExtensibilityKind extensibility_kind** () const

  Gets the extensibility kind.
- size_t **member_count** () const

  Gets the number of members.
- const MemberType & **member** (**MemberIndex** index) const

  Gets a member by its index.
- const MemberType & **member** (const std::string & **name**) const

  Gets a member by its name.
- **MemberIndex find_member_by_name** (const std::string & **name**) const

  Obtains the member index from its name.
- std::vector< MemberType > **members** () const

Gets a copy of all the members.

- uint32_t **cdr_serialized_sample_max_size** (**dds::core::policy::DataRepresentationId** representation_id=DDS_AUTO_DATA_REPRESENTATION) const

  Gets the maximum serialized size of samples of this type.

- uint32_t **cdr_serialized_sample_min_size** (**dds::core::policy::DataRepresentationId** representation_id=DDS_AUTO_DATA_REPRESENTATION) const

  Gets the minimum serialized size of samples of this type.

- uint32_t **cdr_serialized_sample_key_max_size** (**dds::core::policy::DataRepresentationId** representation_id=DDS_AUTO_DATA_REPRESENTATION) const

  Gets the maximum serialized size of sample keys of this type.

Static Public Attributes

- static OMG_DDS_API_CLASS_VARIABLE const **MemberIndex INVALID_INDEX**

  Indicates that a member doesn't exist.

Additional Inherited Members

8.3.1 Detailed Description

template<typename MemberType>
class dds::core::xtypes::AbstractConstructedType< MemberType >

The base class of types that have members and an extensibility kind.

Template Parameters

MemberType	The concrete member type, specified by the subclasses, can be **Member** (p. 1700), **UnionMember** (p. 2578) and **EnumMember** (p. 1533)

This is the base class of **StructType** (p. 2400), **UnionType** (p. 2584) and **EnumType** (p. 1535).

8.3.2 Member Typedef Documentation

8.3.2.1 Member

```
template<typename MemberType >
typedef MemberType  dds::core::xtypes::AbstractConstructedType< MemberType >::Member
```

The member type (specified by the subclasses of this class)

8.3.2.2 MemberIndex

```
template<typename MemberType >
typedef uint32_t  dds::core::xtypes::AbstractConstructedType< MemberType >::MemberIndex
```

The member index type.

8.3.3 Member Function Documentation

8.3.3.1 extensibility_kind()

```
template<typename MemberType >
dds::core::xtypes::ExtensibilityKind  dds::core::xtypes::AbstractConstructedType< MemberType >↩
::extensibility_kind ( ) const
```

Gets the extensibility kind.

8.3.3.2 member_count()

```
template<typename MemberType >
size_t  dds::core::xtypes::AbstractConstructedType< MemberType >::member_count ( ) const
```

Gets the number of members.

8.3.3.3 member() [1/2]

```
template<typename MemberType >
const MemberType &  dds::core::xtypes::AbstractConstructedType< MemberType >::member (
              MemberIndex index ) const
```

Gets a member by its index.

Parameters

index	The index of the member (0 to **member_count()** (p. 787) - 1)

8.3.3.4 member() [2/2]

```
template<typename MemberType >
const MemberType &  dds::core::xtypes::AbstractConstructedType< MemberType >::member (
            const std::string & name ) const
```

Gets a member by its name.

8.3.3.5 find_member_by_name()

```
template<typename MemberType >
MemberIndex  dds::core::xtypes::AbstractConstructedType< MemberType >::find_member_by_name (
            const std::string & name ) const
```

Obtains the member index from its name.

This method is applicable to **dds::core::xtypes::DynamicType** (p. 1504) objects representing structs (**dds::core::xtypes::TypeKind_def::STRUCTURE_TYPE** (p. 2574)) and union (**dds::core::xtypes::TypeKind_def::UNION_TYPE** (p. 2574)) types.

Parameters

name	<<***in***>> *(p. 355)* The member name.

Returns

The index of the member of the given name or **dds::core::xtypes::AbstractConstructedType::INVALID_INDEX** (p. 790) if the member is not found.

MT Safety:

SAFE.

8.3.3.6 members()

```
template<typename MemberType >
std::vector< MemberType >  dds::core::xtypes::AbstractConstructedType< MemberType >::members ( )
const
```

Gets a copy of all the members.

8.3.3.7 cdr_serialized_sample_max_size()

```
template<typename MemberType >
uint32_t  dds::core::xtypes::AbstractConstructedType< MemberType >::cdr_serialized_sample_max_↩
size (
              dds::core::policy::DataRepresentationId representation_id = DDS_AUTO_DATA_REPRESENTATION
) const
```

Gets the maximum serialized size of samples of this type.

Obtains the maximum possible size in bytes of any serialized data sample of this type.

Precondition

The type is an aggregation type (struct, union)

Parameters

representation↩ _id	The serialized data representation for which we calculate the maximum size.

Returns

The maximum size

8.3.3.8 cdr_serialized_sample_min_size()

```
template<typename MemberType >
uint32_t  dds::core::xtypes::AbstractConstructedType< MemberType >::cdr_serialized_sample_min_↩
size (
              dds::core::policy::DataRepresentationId representation_id = DDS_AUTO_DATA_REPRESENTATION
) const
```

Gets the minimum serialized size of samples of this type.

Obtains the minimum possible size in bytes of any serialized data sample of this type.

Precondition

The type is an aggregation type (struct, union)

Parameters

representation↩ _id	The serialized data representation for which we calculate the minimum size.

Returns

The minimum size

8.3.3.9 cdr_serialized_sample_key_max_size()

```
template<typename MemberType >
uint32_t  dds::core::xtypes::AbstractConstructedType< MemberType >::cdr_serialized_sample_key_↩
max_size (
            dds::core::policy::DataRepresentationId representation_id = DDS_AUTO_DATA_REPRESENTATION
) const
```

Gets the maximum serialized size of sample keys of this type.

Obtains the maximum possible size in bytes of the serialized keys of any data sample of this type

Precondition

The type is an aggregation type (struct, union)

Parameters

representation↩ _id	The serialized data representation for which we calculate the maximum key size.

Returns

The maximum key size

8.3.4 Member Data Documentation

8.3.4.1 INVALID_INDEX

```
template<typename MemberType >
OMG_DDS_API_CLASS_VARIABLE const  MemberIndex  dds::core::xtypes::AbstractConstructedType< Member↩
Type >::INVALID_INDEX  [static]
```

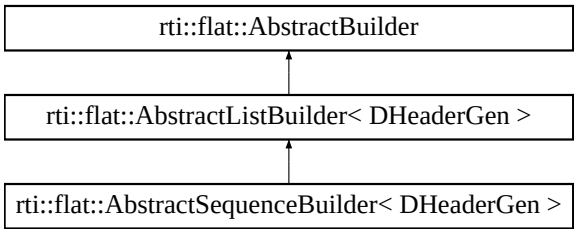
Indicates that a member doesn't exist.

8.4 rti::flat::AbstractListBuilder< DHeaderGen > Class Template Reference

Base class of all array and sequence builders.

`#include <SequenceBuilders.hpp>`

Inheritance diagram for rti::flat::AbstractListBuilder< DHeaderGen >:



Protected Member Functions

- unsigned int **element_count** () const

  Returns the current number of elements that have been added.

Additional Inherited Members

8.4.1 Detailed Description

template<typename DHeaderGen>
class rti::flat::AbstractListBuilder< DHeaderGen >

Base class of all array and sequence builders.

8.4.2 Member Function Documentation

8.4.2.1 element_count()

```
template<typename DHeaderGen >
unsigned int  rti::flat::AbstractListBuilder< DHeaderGen >::element_count ( ) const  [inline],
[protected]
```

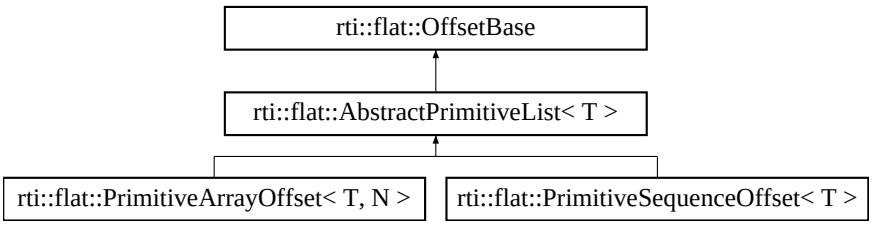
Returns the current number of elements that have been added.

8.5 rti::flat::AbstractPrimitiveList< T > Class Template Reference

Base class for Offsets to sequences and arrays of primitive types.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::AbstractPrimitiveList< T >:



Public Member Functions

- T **get_element** (unsigned int i) const
  Returns an element by index.
- bool **set_element** (unsigned int i, T value)
  Sets an element by index.

8.5.1 Detailed Description

template<typename T>
class rti::flat::AbstractPrimitiveList< T >

Base class for Offsets to sequences and arrays of primitive types.

Template Parameters

T	The primitive type

8.5.2 Member Function Documentation

8.5.2.1 get_element()

```
template<typename T >
T  rti::flat::AbstractPrimitiveList< T >::get_element (
            unsigned int i ) const  [inline]
```

Returns an element by index.

Parameters

i	The zero-based index of the element

See also

rti::flat::plain_cast() (p. 417) for a method to access **all** (p. 699) the elements at once

8.5.2.2 set_element()

```
template<typename T >
bool  rti::flat::AbstractPrimitiveList< T >::set_element (
            unsigned int i,
            T value )  [inline]
```

Sets an element by index.

Parameters

i	The zero-based index of the element to set
value	The value to set

Returns

true if it was possible to set the element, or false if this collection has less than i - 1 elements.

See also

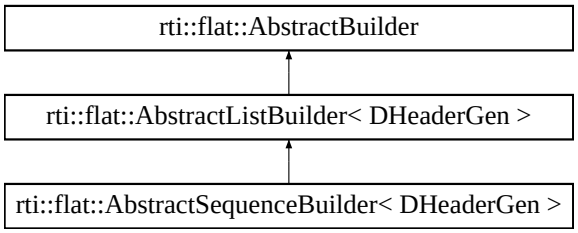
rti::flat::plain_cast() (p. 417) for a method to access **all** (p. 699) the elements at once

8.6 rti::flat::AbstractSequenceBuilder< DHeaderGen > Class Template Reference

Base class of Builders for sequence members.

`#include <SequenceBuilders.hpp>`

Inheritance diagram for rti::flat::AbstractSequenceBuilder< DHeaderGen >:



Public Member Functions

- unsigned int **element_count** () const

  Returns the current number of elements that have been added.

Additional Inherited Members

8.6.1 Detailed Description

template<typename DHeaderGen>
class rti::flat::AbstractSequenceBuilder< DHeaderGen >

Base class of Builders for sequence members.

This class contains only implementation details and doesn't add any public function to **AbstractListBuilder** (p. 791).

8.6.2 Member Function Documentation

8.6.2.1 element_count()

```
template<typename DHeaderGen >
unsigned int  rti::flat::AbstractListBuilder< DHeaderGen >::element_count ( ) const  [inline]
```

Returns the current number of elements that have been added.

8.7 rti::pub::AcknowledgmentInfo Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Information about an application-acknowledged sample

```
#include <AcknowledgmentInfo.hpp>
```

Public Member Functions

- **dds::core::InstanceHandle subscription_handle** () const

  *Gets the subscription handle of the acknowledging **dds::sub::DataReader** (p. 976).*
- **rti::core::SampleIdentity sample_identity** () const

  Gets the identity of the sample being acknowledged.
- bool **valid_response_data** () const

  Flag indicating validity of the user response data in the acknowledgment.
- **rti::sub::AckResponseData response_data** () const

  User data payload of application-level acknowledgment message.

8.7.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Information about an application-acknowledged sample

When acknowledging a sample, the reader provides the writer with information about the sample being acknowledged. The **AcknowledgmentInfo** (p. 795) structure provides the identity and cookie of the sample being acknowledged, as well as user data payload provided by the reader.

8.7.2 Member Function Documentation

8.7.2.1 subscription_handle()

```
dds::core::InstanceHandle rti::pub::AcknowledgmentInfo::subscription_handle ( ) const
```

Gets the subscription handle of the acknowledging **dds::sub::DataReader** (p. 976).

8.7.2.2 sample_identity()

```
rti::core::SampleIdentity rti::pub::AcknowledgmentInfo::sample_identity ( ) const
```

Gets the identity of the sample being acknowledged.

See also

rti::core::SampleIdentity (p. 2282)

8.7.2.3 valid_response_data()

```
bool rti::pub::AcknowledgmentInfo::valid_response_data ( ) const
```

Flag indicating validity of the user response data in the acknowledgment.

This flag is true when the **rti::core::RtpsReliableReaderProtocol::min_app_ack_response_keep_duration** (p. 2227) has not yet elapsed for the acknowledgment's response data.

The flag is false when that duration has elapsed for the response data.

8.7.2.4 response_data()

```
rti::sub::AckResponseData rti::pub::AcknowledgmentInfo::response_data ( ) const
```

User data payload of application-level acknowledgment message.

Response data set by **dds::sub::DataReader** (p. 976) when sample was acknowledged.

8.8 rti::core::policy::AcknowledgmentKind_def Struct Reference

<<***extension***>> *(p. 354)* The enumeration for Reliability acknowledgment kinds

`#include <PolicyKind.hpp>`

Public Types

- enum **type** {
  PROTOCOL ,
  APPLICATION_AUTO ,
  APPLICATION_ORDERED ,
  APPLICATION_EXPLICIT }
  The underlying `enum` type.

8.8.1 Detailed Description

<<***extension***>> *(p. 354)* The enumeration for Reliability acknowledgment kinds

8.8.2 Member Enumeration Documentation

8.8.2.1 type

```
enum rti::core::policy::AcknowledgmentKind_def::type
```

The underlying `enum` type.

Enumerator

PROTOCOL	Samples are acknowledged by RTPS protocol. Samples are acknowledged according to the Real-Time Publish-Subscribe (RTPS) interoperability protocol.

Enumerator

APPLICATION_AUTO	Samples are acknowledged automatically after a subscribing application has accessed them. The DataReader automatically acknowledges a sample after it has been taken and the loan returned. **See also** **dds::sub::DataReader::take()** (p. 997) **dds::sub::LoanedSamples::~LoanedSamples()** (p. 1666).
APPLICATION_ORDERED	Samples up to a specified sequence number are acknowledged.
APPLICATION_EXPLICIT	Samples are acknowledged after the subscribing application explicitly calls acknowledge on the samples. Samples received by a **dds::sub::DataReader** (p. 976) are explicitly acknowledged by the subscribing application, after it calls either **dds::sub::DataReader::acknowledge_all** (p. 1013) or **dds::sub::DataReader::acknowledge_sample** (p. 1015).

8.9 rti::sub::AckResponseData Class Reference

<<***extension***>> *(p. 354)* Data payload associated to an application-level acknowledgment

`#include <rti/sub/AckResponseData.hpp>`

Public Member Functions

- **AckResponseData** ()
  Creates an empty sequence of bytes.
- **AckResponseData** (const **dds::core::ByteSeq** &sequence)
  Creates an instance by copying a sequence of bytes.
- **dds::core::ByteSeq value** () const
  Gets the sequence of bytes.
- **dds::core::ByteSeq** & **value** (**dds::core::ByteSeq** &dst) const
  Sets the sequence of bytes.
- const uint8_t * **begin** () const
  Provides access to the beginning of the sequence of bytes.
- const uint8_t * **end** () const
  Gets the end of the sequence.

8.9.1 Detailed Description

<<***extension***>> *(p. 354)* Data payload associated to an application-level acknowledgment

When a **dds::sub::DataReader** (p. 976) explicitly acknowledges samples with **dds::sub::DataReader::acknowledge_sample** (p. 1015) or **dds::sub::DataReader::acknowledge_all** (p. 1013), it may specify opaque data to send as payload in the acknowledgment message sent to the **dds::pub::DataWriter** (p. 1138).

8.9.2 Constructor & Destructor Documentation

8.9.2.1 AckResponseData() [1/2]

```
rti::sub::AckResponseData::AckResponseData ( )  [inline]
```

Creates an empty sequence of bytes.

8.9.2.2 AckResponseData() [2/2]

```
rti::sub::AckResponseData::AckResponseData (
            const  dds::core::ByteSeq & sequence )  [inline], [explicit]
```

Creates an instance by copying a sequence of bytes.

8.9.3 Member Function Documentation

8.9.3.1 value() [1/2]

```
dds::core::ByteSeq rti::sub::AckResponseData::value ( ) const  [inline]
```

Gets the sequence of bytes.

8.9.3.2 value() [2/2]

```
dds::core::ByteSeq & rti::sub::AckResponseData::value (
             dds::core::ByteSeq & dst ) const
```

Sets the sequence of bytes.

8.9.3.3 begin()

```
const uint8_t * rti::sub::AckResponseData::begin ( ) const
```

Provides access to the beginning of the sequence of bytes.

8.9.3.4 end()

```
const uint8_t * rti::sub::AckResponseData::end ( ) const
```

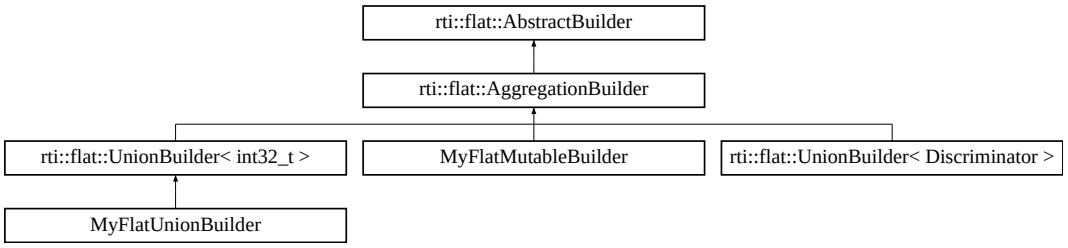
Gets the end of the sequence.

8.10 rti::flat::AggregationBuilder Class Reference

Base class of struct and union builders.

```
#include <AggregationBuilders.hpp>
```

Inheritance diagram for rti::flat::AggregationBuilder:



Additional Inherited Members

8.10.1 Detailed Description

Base class of struct and union builders.

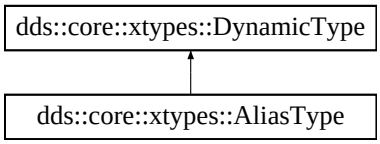
This class contains implementation details and doesn't add any public function to **AbstractBuilder** (p. 783). See **My↩ FlatMutableBuilder** (p. 1763) for a concrete example of a struct builder.

8.11 dds::core::xtypes::AliasType Class Reference

<<***value-type***>> *(*p. *350)* Represents an IDL `typedef`

`#include <dds/core/xtypes/AliasType.hpp>`

Inheritance diagram for dds::core::xtypes::AliasType:



Public Member Functions

- **AliasType** (const std::string & **name**, const DynamicTypeImpl & **related_type**, bool **is_pointer**=false)
  Creates an alias with a name and a related type.
- const **dds::core::xtypes::DynamicType** & **related_type** () const
  Gets the related type.
- bool **is_pointer** () const
  <<***extension***>> *(*p. *354) Gets whether this alias makes related_type a pointer.*
- **dds::core::optional**< std::string > **unit** () const
  <<***extension***>> *(*p. *354) Gets a copy of the unit annotations of this alias.*

Related Functions

(Note that these are not member functions.)

- const **dds::core::xtypes::DynamicType** & **resolve_alias** (const **dds::core::xtypes::DynamicType** &type)
  <<***extension***>> *(*p. *354) If the type is an alias returns its related type recursively until it is not an alias.*

8.11.1 Detailed Description

<<***value-type***>> *(*p. *350)* Represents an IDL `typedef`

8.11.2 Constructor & Destructor Documentation

8.11.2.1 AliasType()

```
dds::core::xtypes::AliasType::AliasType (
            const std::string & name,
            const DynamicTypeImpl & related_type,
            bool is_pointer = false )
```

Creates an alias with a name and a related type.

Parameters

name	The name of this alias
related_type	The aliased type
is_pointer	Whether this alias makes the type a pointer

For example, to create the following two IDL aliases:

```
typedef long my_long;
typedef long * my_long_ptr;
```

We can use the following code:

```
AliasType my_long("my_long", primitive_type<int32_t>());
AliasType my_long_ptr("my_long_ptr", primitive_type<int32_t>(), true);
```

8.11.3 Member Function Documentation

8.11.3.1 related_type()

```
const  dds::core::xtypes::DynamicType & dds::core::xtypes::AliasType::related_type ( ) const
```

Gets the related type.

8.11.3.2 is_pointer()

```
bool dds::core::xtypes::AliasType::is_pointer ( ) const
```

<<***extension***>> *(*p. *354)* Gets whether this alias makes related_type a pointer.

8.11.3.3 unit()

```
dds::core::optional< std::string > dds::core::xtypes::AliasType::unit ( ) const
```

<<***extension***>> *(*p. *354)* Gets a copy of the unit annotations of this alias.

8.11.4 Friends And Related Function Documentation

8.11.4.1 resolve_alias()

```
const  dds::core::xtypes::DynamicType & resolve_alias (
             const  dds::core::xtypes::DynamicType & type )  [related]
```

<<***extension***>> *(p. 354)* If the type is an alias returns its related type recursively until it is not an alias.

8.12 rti::core::AllocationSettings Class Reference

<<***extension***>> *(p. 354)* Resource allocation settings

`#include <rti/core/PolicySettings.hpp>`

Public Member Functions

- **AllocationSettings** ()
  Creates an instance with an initial, max and incremental count set to zero.
- **AllocationSettings** (int32_t the_initial_count, int32_t the_max_count, int32_t the_incremental_count)
  Creates an instance with the given initial, maximum and incremental values.
- int32_t **initial_count** () const
  Getter (see setter with the same name)
- **AllocationSettings** & **initial_count** (int32_t the_initial_count)
  Sets the initial count of resources.
- int32_t **max_count** () const
  Getter (see setter with the same name)
- **AllocationSettings** & **max_count** (int32_t the_max_count)
  Sets the maximum count of resources.
- int32_t **incremental_count** () const
  Getter (see setter with the same name)
- **AllocationSettings** & **incremental_count** (int32_t the_incremental_count)
  Sets the incremental count of resources.

Static Public Attributes

- static const int32_t **AUTO_COUNT** = -2
  A special value indicating that the quantity is derived from a different Qos value.

8.12.1 Detailed Description

<<***extension***>> *(p. 354)* Resource allocation settings

QoS:

rti::core::policy::DomainParticipantResourceLimits (p. 1391)

8.12.2 Constructor & Destructor Documentation

8.12.2.1 AllocationSettings() [1/2]

```
rti::core::AllocationSettings::AllocationSettings ( )  [inline]
```

Creates an instance with an initial, max and incremental count set to zero.

8.12.2.2 AllocationSettings() [2/2]

```
rti::core::AllocationSettings::AllocationSettings (
            int32_t the_initial_count,
            int32_t the_max_count,
            int32_t the_incremental_count )  [inline]
```

Creates an instance with the given initial, maximum and incremental values.

8.12.3 Member Function Documentation

8.12.3.1 initial_count() [1/2]

```
int32_t rti::core::AllocationSettings::initial_count ( ) const  [inline]
```

Getter (see setter with the same name)

8.12.3.2 initial_count() [2/2]

```
AllocationSettings & rti::core::AllocationSettings::initial_count (
            int32_t the_initial_count )  [inline]
```

Sets the initial count of resources.

The initial resources to be allocated.

[default] It depends on the case.

[range] [0, 1 million], <= max_count

8.12.3.3 max_count() [1/2]

```
int32_t rti::core::AllocationSettings::max_count ( ) const  [inline]
```

Getter (see setter with the same name)

8.12.3.4 max_count() [2/2]

```
AllocationSettings & rti::core::AllocationSettings::max_count (
            int32_t the_max_count )  [inline]
```

Sets the maximum count of resources.

The maximum resources to be allocated.

[default] Depends on the case.

[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_count

8.12.3.5 incremental_count() [1/2]

```
int32_t rti::core::AllocationSettings::incremental_count ( ) const  [inline]
```

Getter (see setter with the same name)

8.12.3.6 incremental_count() [2/2]

```
AllocationSettings & rti::core::AllocationSettings::incremental_count (
            int32_t the_incremental_count )  [inline]
```

Sets the incremental count of resources.

The resource to be allocated when more resources are needed.

[default] Depends on the case.

[range] -1 (Double the amount of extra memory allocated each time memory is needed) or [1,1 million] (or = 0 only if initial_count == max_count)

8.12.4 Member Data Documentation

8.12.4.1 AUTO_COUNT

```
const int32_t rti::core::AllocationSettings::AUTO_COUNT = -2  [static]
```

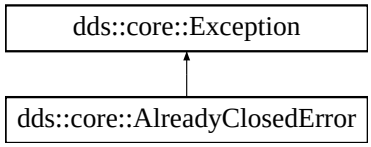
A special value indicating that the quantity is derived from a different Qos value.

8.13 dds::core::AlreadyClosedError Class Reference

Indicates that an object has been closed.

```
#include <Exception.hpp>
```

Inheritance diagram for dds::core::AlreadyClosedError:



Public Member Functions

- virtual const char * **what** () const throw ()

  *Access the message contained in this **AlreadyClosedError** (p. 806) exception.*

8.13.1 Detailed Description

Indicates that an object has been closed.

Inherits also from `std::logic_error`

See also

Tearing Down An Entity (p. 328)

8.13.2 Member Function Documentation

8.13.2.1 what()

```
virtual const char * dds::core::AlreadyClosedError::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **AlreadyClosedError** (p. 806) exception.

Returns

The message.

Implements **dds::core::Exception** (p. 1547).

8.14 dds::sub::AnyDataReader Class Reference

<<***reference-type***>> *(p. 351)* This class provides an non-template holder for representing a **DataReader** (p. 976) of any type

```
#include "dds/sub/AnyDataReader.hpp"
```

Public Member Functions

- template<typename T >
  AnyDataReader (const **dds::sub::DataReader**< T > &reader)
  *Create an **AnyDataReader** (p. 807) that represents a generic, untyped **DataReader** (p. 976).*
- **AnyDataReader** & **operator=** (const **AnyDataReader** &other)
  Copy assignment operator.
- const **dds::sub::qos::DataReaderQos** **qos** () const
  *Get the QoS for this **AnyDataReader** (p. 807).*
- void **qos** (const **dds::sub::qos::DataReaderQos** &the_qos)
  *Set the QoS for this **AnyDataReader** (p. 807).*
- const std::string & **topic_name** () const
  *Get the topic name for this **AnyDataReader** (p. 807).*
- const std::string & **type_name** () const
  *Get the type name for this **AnyDataReader** (p. 807).*
- const **dds::sub::Subscriber** & **subscriber** () const
  *Get the publisher for this **AnyDataReader** (p. 807).*
- template<typename T >
  AnyDataReader & **operator=** (const **DataReader**< T > &other)
  *Assign a different **DataReader** (p. 976).*
- template<typename T >
  dds::sub::DataReader< T > **get** () const
  *Gets the typed **DataReader** (p. 976) from this **AnyDataReader** (p. 807).*
- void **close** ()
  *Close this **AnyDataReader** (p. 807).*
- void **retain** ()
  *Retain this **AnyDataReader** (p. 807).*

Related Functions

(Note that these are not member functions.)

- template<typename T >
  DataReader< T > **get** (const **AnyDataReader** &any_reader)

  *Same as **AnyDataReader::get()** (p. 810)*

8.14.1 Detailed Description

<<***reference-type***>> *(p. 351)* This class provides an non-template holder for representing a **DataReader** (p. 976) of any type

This class is useful for code that uses DataReaders of different types. It provides access to type-independent **DataReader** (p. 976) functions.

An **AnyDataReader** (p. 807) is always created from a typed **DataReader** (p. 976), increasing its reference count. The member function **get()** (p. 810) allows getting the typed **DataReader** (p. 976) back.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 AnyDataReader()

```
template<typename T >
dds::sub::AnyDataReader::AnyDataReader (
            const  dds::sub::DataReader< T > & reader )  [inline]
```

Create an **AnyDataReader** (p. 807) that represents a generic, untyped **DataReader** (p. 976).

Template Parameters

T	The type of the **DataReader** (p. 976) that this **AnyDataReader** (p. 807) is being created from

Parameters

reader	The typed **DataReader** (p. 976) that this **AnyDataReader** (p. 807) will represent

Referenced by **operator=()**.

8.14.3 Member Function Documentation

8.14.3.1 operator=() [1/2]

```
AnyDataReader & dds::sub::AnyDataReader::operator= (
            const  AnyDataReader & other )  [inline]
```

Copy assignment operator.

MT Safety:

UNSAFE. It is not safe to call the assignment while another thread may be calling other methods on this object.

8.14.3.2 qos() [1/2]

```
const  dds::sub::qos::DataReaderQos dds::sub::AnyDataReader::qos ( ) const  [inline]
```

Get the QoS for this **AnyDataReader** (p. 807).

Returns

dds::sub::qos::DataReaderQos (p. 1072) The DataReaderQos

8.14.3.3 qos() [2/2]

```
void dds::sub::AnyDataReader::qos (
            const  dds::sub::qos::DataReaderQos & the_qos )  [inline]
```

Set the QoS for this **AnyDataReader** (p. 807).

Parameters

the_qos	The QoS to set

8.14.3.4 topic_name()

```
const std::string & dds::sub::AnyDataReader::topic_name ( ) const  [inline]
```

Get the topic name for this **AnyDataReader** (p. 807).

Returns

The topic name

8.14.3.5 type_name()

```
const std::string & dds::sub::AnyDataReader::type_name ( ) const  [inline]
```

Get the type name for this **AnyDataReader** (p. 807).

Returns

The type name

8.14.3.6 subscriber()

```
const  dds::sub::Subscriber & dds::sub::AnyDataReader::subscriber ( ) const  [inline]
```

Get the publisher for this **AnyDataReader** (p. 807).

Returns

The subscriber

8.14.3.7 operator=() [2/2]

```
template<typename T >
AnyDataReader & dds::sub::AnyDataReader::operator= (
            const  DataReader< T > & other )  [inline]
```

Assign a different **DataReader** (p. 976).

Replaces the reference to the current reader with a new one, whose type `T` can be different.

MT Safety:

UNSAFE. It is not safe to call the assignment while another thread may be calling other methods on this object.

References **AnyDataReader()**.

8.14.3.8 get()

```
template<typename T >
dds::sub::DataReader< T > dds::sub::AnyDataReader::get ( ) const  [inline]
```

Gets the typed **DataReader** (p. 976) from this **AnyDataReader** (p. 807).

Template Parameters

T	The type of the **DataReader** (p. 976) to extract

Returns

The typed **DataReader** (p. 976) that this **AnyDataReader** (p. 807) represents

Exceptions

dds::core::InvalidDowncastError *(p. 1620)*	If this **AnyDataReader** (p. 807) doesn't represent a DataReader<T>.

MT Safety:

UNSAFE. It is not safe to call **get<T>()** (p. 810) while another thread may be calling other methods on this object.

Referenced by **get()**.

8.14.3.9 close()

```
void dds::sub::AnyDataReader::close ( )  [inline]
```

Close this **AnyDataReader** (p. 807).

See also

DataReader::close() (p. 1024)

8.14.3.10 retain()

```
void dds::sub::AnyDataReader::retain ( )  [inline]
```

Retain this **AnyDataReader** (p. 807).

See also

DataReader::retain() (p. 1526)

8.14.4 Friends And Related Function Documentation

8.14.4.1 get()

```
template<typename T >
DataReader< T > get (
            const  AnyDataReader & any_reader )  [related]
```

Same as **AnyDataReader::get()** (p. 810)

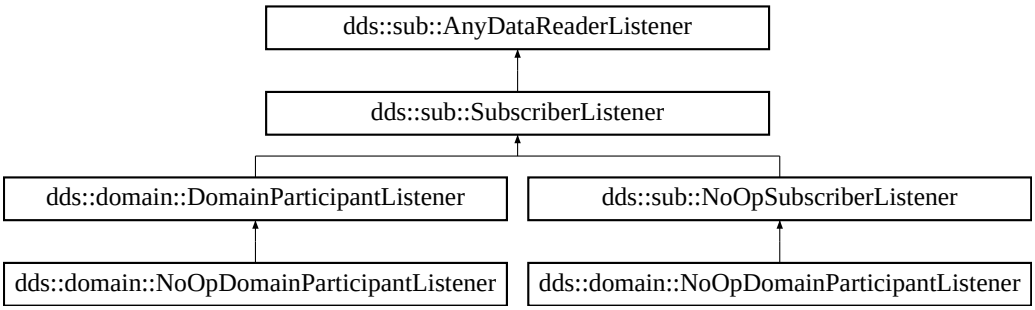
References **get()**.

8.15 dds::sub::AnyDataReaderListener Class Reference

The listener to notify status changes for a **dds::sub::DataReader** (p. 976) of a generic type.

`#include <dds/sub/AnyDataReaderListener.hpp>`

Inheritance diagram for dds::sub::AnyDataReaderListener:



Public Member Functions

- virtual void **on_requested_deadline_missed** (**AnyDataReader** &the_reader, const **dds::core::status::RequestedDeadlineMissedStatus** &status)=0

  *Handles the **dds::core::status::RequestedDeadlineMissedStatus** (p. 2184) status.*
- virtual void **on_requested_incompatible_qos** (**AnyDataReader** &the_reader, const **dds::core::status::RequestedIncompatibleQosStatus** &status)=0

  *Handles the **dds::core::status::RequestedIncompatibleQosStatus** (p. 2185) status.*
- virtual void **on_sample_rejected** (**AnyDataReader** &the_reader, const **dds::core::status::SampleRejectedStatus** &status)=0

  *Handles the **dds::core::status::SampleRejectedStatus** (p. 2315) status.*
- virtual void **on_liveliness_changed** (**AnyDataReader** &the_reader, const **dds::core::status::LivelinessChangedStatus** &status)=0

  *Handles the **dds::core::status::LivelinessChangedStatus** (p. 1652) status.*
- virtual void **on_data_available** (**AnyDataReader** &the_reader)=0

  Handles the dds::core::status::DataAvailableStatus status.
- virtual void **on_subscription_matched** (**AnyDataReader** &the_reader, const **dds::core::status::SubscriptionMatchedStatus** &status)=0

  *Handles the **dds::core::status::SubscriptionMatchedStatus** (p. 2439) status.*
- virtual void **on_sample_lost** (**AnyDataReader** &the_reader, const **dds::core::status::SampleLostStatus** &status)=0

  *Handles the **dds::core::status::SampleLostStatus** (p. 2303) status.*

8.15.1 Detailed Description

The listener to notify status changes for a **dds::sub::DataReader** (p. 976) of a generic type.

This class is only intended to act as the base of **SubscriberListener** (p. 2422).

See also

DataReaderListener (p. 1055)

SubscriberListener (p. 2422)

8.15.2 Member Function Documentation

8.15.2.1 on_requested_deadline_missed()

```
virtual void dds::sub::AnyDataReaderListener::on_requested_deadline_missed (
              AnyDataReader & the_reader,
             const  dds::core::status::RequestedDeadlineMissedStatus & status )  [pure virtual]
```

Handles the **dds::core::status::RequestedDeadlineMissedStatus** (p. 2184) status.

Implemented in **dds::sub::NoOpSubscriberListener** (p. 1865), and **dds::domain::NoOpDomainParticipant↩ Listener** (p. 1850).

8.15.2.2 on_requested_incompatible_qos()

```
virtual void dds::sub::AnyDataReaderListener::on_requested_incompatible_qos (
              AnyDataReader & the_reader,
             const  dds::core::status::RequestedIncompatibleQosStatus & status )  [pure virtual]
```

Handles the **dds::core::status::RequestedIncompatibleQosStatus** (p. 2185) status.

Implemented in **dds::sub::NoOpSubscriberListener** (p. 1865), and **dds::domain::NoOpDomainParticipant↩ Listener** (p. 1850).

8.15.2.3 on_sample_rejected()

```
virtual void dds::sub::AnyDataReaderListener::on_sample_rejected (
             AnyDataReader & the_reader,
            const  dds::core::status::SampleRejectedStatus & status )  [pure virtual]
```

Handles the **dds::core::status::SampleRejectedStatus** (p. 2315) status.

Implemented in **dds::sub::NoOpSubscriberListener** (p. 1865), and **dds::domain::NoOpDomainParticipant↩ Listener** (p. 1850).

8.15.2.4 on_liveliness_changed()

```
virtual void dds::sub::AnyDataReaderListener::on_liveliness_changed (
             AnyDataReader & the_reader,
            const  dds::core::status::LivelinessChangedStatus & status )  [pure virtual]
```

Handles the **dds::core::status::LivelinessChangedStatus** (p. 1652) status.

Implemented in **dds::sub::NoOpSubscriberListener** (p. 1866), and **dds::domain::NoOpDomainParticipant↩ Listener** (p. 1851).

8.15.2.5 on_data_available()

```
virtual void dds::sub::AnyDataReaderListener::on_data_available (
             AnyDataReader & the_reader )  [pure virtual]
```

Handles the dds::core::status::DataAvailableStatus status.

Implemented in **dds::sub::NoOpSubscriberListener** (p. 1866), and **dds::domain::NoOpDomainParticipant↩ Listener** (p. 1851).

8.15.2.6 on_subscription_matched()

```
virtual void dds::sub::AnyDataReaderListener::on_subscription_matched (
             AnyDataReader & the_reader,
            const  dds::core::status::SubscriptionMatchedStatus & status )  [pure virtual]
```

Handles the **dds::core::status::SubscriptionMatchedStatus** (p. 2439) status.

Implemented in **dds::sub::NoOpSubscriberListener** (p. 1866), and **dds::domain::NoOpDomainParticipant↩ Listener** (p. 1851).

8.15.2.7 on_sample_lost()

```
virtual void dds::sub::AnyDataReaderListener::on_sample_lost (
              AnyDataReader & the_reader,
             const  dds::core::status::SampleLostStatus & status )  [pure virtual]
```

Handles the **dds::core::status::SampleLostStatus** (p. 2303) status.

Implemented in **dds::sub::NoOpSubscriberListener** (p. 1866), and **dds::domain::NoOpDomainParticipant↩ Listener** (p. 1851).

8.16 dds::pub::AnyDataWriter Class Reference

<<***reference-type***>> *(p. 351)* This class provides an non-template holder for representing a **DataWriter** (p. 1138) of any type

```
#include "dds/pub/AnyDataWriter.hpp"
```

Public Member Functions

- template<typename T >
  AnyDataWriter (const **dds::pub::DataWriter**< T > &writer)

  *Create an **AnyDataWriter** (p. 815) that represents a generic, untyped **DataWriter** (p. 1138).*
- **AnyDataWriter** & **operator=** (const **AnyDataWriter** &other)

  Copy assignment operator.
- const **dds::pub::qos::DataWriterQos** **qos** () const

  *Get the QoS for this **AnyDataWriter** (p. 815).*
- void **qos** (const **dds::pub::qos::DataWriterQos** &the_qos)

  *Set the QoS for this **AnyDataWriter** (p. 815).*
- const std::string & **topic_name** () const

  *Get the topic name for this **AnyDataWriter** (p. 815).*
- const std::string & **type_name** () const

  *Get the type name for this **AnyDataWriter** (p. 815).*
- const **dds::pub::Publisher** & **publisher** () const

  *Get the publisher for this **AnyDataWriter** (p. 815).*
- void **wait_for_acknowledgments** (const **dds::core::Duration** &timeout)
- void **close** ()

  *Close the **DataWriter** (p. 1138).*
- void **retain** ()

  *Retain this **AnyDataWriter** (p. 815).*
- template<typename T >
  AnyDataWriter & **operator=** (const **dds::pub::DataWriter**< T > &other)

  *Assign a different **DataWriter** (p. 1138).*
- template<typename T >
  DataWriter< T > **get** ()

  *Gets the typed **DataWriter** (p. 1138) from this **AnyDataWriter** (p. 815).*

Related Functions

(Note that these are not member functions.)

- template<typename T >
  DataWriter< T > **get** (const **AnyDataWriter** &any_writer)

  *Same as **AnyDataWriter::get()** (p. 819)*

8.16.1 Detailed Description

<<***reference-type***>> *(p. 351)* This class provides an non-template holder for representing a **DataWriter** (p. 1138) of any type

This class is useful for code that uses DataWriters of different types. It provides access to type-independent **DataWriter** (p. 1138) functions.

An **AnyDataWriter** (p. 815) is always created from a typed **DataWriter** (p. 1138), increasing its reference count. The member function **get()** (p. 819) allows getting the typed **DataWriter** (p. 1138) back.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 AnyDataWriter()

```
template<typename T >
dds::pub::AnyDataWriter::AnyDataWriter (
            const  dds::pub::DataWriter< T > & writer )  [inline]
```

Create an **AnyDataWriter** (p. 815) that represents a generic, untyped **DataWriter** (p. 1138).

Template Parameters

T	The type of the **DataWriter** (p. 1138) that this **AnyDataWriter** (p. 815) is being created from

Parameters

writer	The typed **DataWriter** (p. 1138) that this **AnyDataWriter** (p. 815) will represent

Referenced by **operator=()**.

8.16.3 Member Function Documentation

8.16.3.1 operator=() [1/2]

```
AnyDataWriter & dds::pub::AnyDataWriter::operator= (
            const  AnyDataWriter & other )  [inline]
```

Copy assignment operator.

MT Safety:

UNSAFE. It is not safe to call the assignment while another thread may be calling other methods on this object.

8.16.3.2 qos() [1/2]

```
const  dds::pub::qos::DataWriterQos dds::pub::AnyDataWriter::qos ( ) const  [inline]
```

Get the QoS for this **AnyDataWriter** (p. 815).

Returns

dds::pub::qos::DataWriterQos (p. 1226) The DataWriterQos

8.16.3.3 qos() [2/2]

```
void dds::pub::AnyDataWriter::qos (
            const  dds::pub::qos::DataWriterQos & the_qos )  [inline]
```

Set the QoS for this **AnyDataWriter** (p. 815).

Parameters

the_qos	The QoS to set

8.16.3.4 topic_name()

```
const std::string & dds::pub::AnyDataWriter::topic_name ( ) const  [inline]
```

Get the topic name for this **AnyDataWriter** (p. 815).

Returns

The topic name

8.16.3.5 type_name()

```
const std::string & dds::pub::AnyDataWriter::type_name ( ) const  [inline]
```

Get the type name for this **AnyDataWriter** (p. 815).

Returns

The type name

8.16.3.6 publisher()

```
const  dds::pub::Publisher & dds::pub::AnyDataWriter::publisher ( ) const  [inline]
```

Get the publisher for this **AnyDataWriter** (p. 815).

Returns

The publisher

8.16.3.7 wait_for_acknowledgments()

```
void dds::pub::AnyDataWriter::wait_for_acknowledgments (
            const  dds::core::Duration & timeout )  [inline]
```

See also

DataWriter::wait_for_acknowledgments (p. 1168)

8.16.3.8 close()

```
void dds::pub::AnyDataWriter::close ( )  [inline]
```

Close the **DataWriter** (p. 1138).

See also

DataWriter::close() (p. 1525)

8.16.3.9 retain()

```
void dds::pub::AnyDataWriter::retain ( )  [inline]
```

Retain this **AnyDataWriter** (p. 815).

See also

DataWriter::retain() (p. 1526)

8.16.3.10 operator=() [2/2]

```
template<typename T >
AnyDataWriter & dds::pub::AnyDataWriter::operator= (
            const  dds::pub::DataWriter< T > & other )  [inline]
```

Assign a different **DataWriter** (p. 1138).

Replaces the reference to the current writer with a new one, whose type `T` can be different.

MT Safety:

UNSAFE. It is not safe to call the assignment while another thread may be calling other methods on this object.

References **AnyDataWriter()**.

8.16.3.11 get()

```
template<typename T >
DataWriter< T > dds::pub::AnyDataWriter::get ( )  [inline]
```

Gets the typed **DataWriter** (p. 1138) from this **AnyDataWriter** (p. 815).

Template Parameters

T	The type of the **DataWriter** (p. 1138) to extract

Returns

The typed **DataWriter** (p. 1138) that this **AnyDataWriter** (p. 815) represents

Exceptions

dds::core::InvalidDowncastError *(p. 1620)*	If this **AnyDataWriter** (p. 815) doesn't represent a DataWriter<T>.

MT Safety:

UNSAFE. It is not safe to call **get<T>()** (p. 819) while another thread may be calling other methods on this object.

Referenced by **get()**.

8.16.4 Friends And Related Function Documentation

8.16.4.1 get()

```
template<typename T >
DataWriter< T > get (
            const  AnyDataWriter & any_writer )  [related]
```

Same as **AnyDataWriter::get()** (p. 819)

Examples

Foo.hpp.

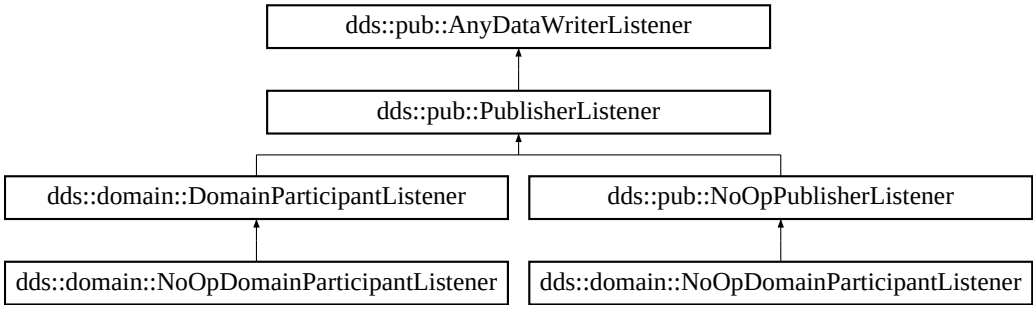
References **get()**.

8.17 dds::pub::AnyDataWriterListener Class Reference

The listener to notify status changes for a **dds::pub::DataWriter** (p. 1138) of a generic type.

`#include <dds/pub/AnyDataWriterListener.hpp>`

Inheritance diagram for dds::pub::AnyDataWriterListener:



Public Member Functions

- virtual void **on_offered_deadline_missed** (**dds::pub::AnyDataWriter** &writer, const **::dds::core::status::↩ OfferedDeadlineMissedStatus** &status)=0

  *Handles the **dds::core::status::OfferedDeadlineMissedStatus** (p. 1871) status.*
- virtual void **on_offered_incompatible_qos** (**dds::pub::AnyDataWriter** &writer, const **::dds::core::status::↩ OfferedIncompatibleQosStatus** &status)=0

  *Handles the **dds::core::status::OfferedIncompatibleQosStatus** (p. 1873) status.*
- virtual void **on_liveliness_lost** (**dds::pub::AnyDataWriter** &writer, const **::dds::core::status::Liveliness↩ LostStatus** &status)=0

  *Handles the **dds::core::status::LivelinessLostStatus** (p. 1655) status.*
- virtual void **on_publication_matched** (**dds::pub::AnyDataWriter** &writer, const **::dds::core::status::↩ PublicationMatchedStatus** &status)=0

  *Handles the **dds::core::status::PublicationMatchedStatus** (p. 1990) status.*
- virtual void **on_reliable_writer_cache_changed** (**dds::pub::AnyDataWriter** &writer, const **rti::core::status↩ ::ReliableWriterCacheChangedStatus** &status)=0

  *<<**extension**>> (p. 354) Handles the dds::core::status::ReliableWriterCacheChangedStatus status*
- virtual void **on_reliable_reader_activity_changed** (**dds::pub::AnyDataWriter** &writer, const **rti::core↩ ::status::ReliableReaderActivityChangedStatus** &status)=0

  *<<**extension**>> (p. 354) Handles the dds::core::status::ReliableReaderActivityChangedStatus status*
- virtual void **on_sample_removed** (**dds::pub::AnyDataWriter** &writer, const **rti::core::Cookie** &cookie)=0

  *<<**extension**>> (p. 354) Notifies when a sample is removed from **DataWriter** (p. 1138) queue.*
- virtual void **on_instance_replaced** (**dds::pub::AnyDataWriter** &writer, const **dds::core::InstanceHandle** &handle)=0

  *<<**extension**>> (p. 354) Notifies when an instance is replaced in **DataWriter** (p. 1138) queue.*
- virtual void **on_application_acknowledgment** (**dds::pub::AnyDataWriter** &writer, const **rti::pub::↩ AcknowledgmentInfo** &acknowledgment_info)=0

  *<<**extension**>> (p. 354) Called when a sample is application-acknowledged.*
- virtual void **on_service_request_accepted** (**dds::pub::AnyDataWriter** &writer, const **rti::core::status::↩ ServiceRequestAcceptedStatus** &status)=0

  *<<**extension**>> (p. 354) Called when a sample that has been received on a the built-in service channel is intended for the **DataWriter** (p. 1138) that has installed this listener*

8.17.1 Detailed Description

The listener to notify status changes for a **dds::pub::DataWriter** (p. 1138) of a generic type.

This class is only intended to act as the base of **PublisherListener** (p. 2006).

See also

DataWriterListener (p. 1202)

PublisherListener (p. 2006)

8.17.2 Member Function Documentation

8.17.2.1 on_offered_deadline_missed()

```
virtual void dds::pub::AnyDataWriterListener::on_offered_deadline_missed (
             dds::pub::AnyDataWriter & writer,
            const  ::dds::core::status::OfferedDeadlineMissedStatus & status )  [pure virtual]
```

Handles the **dds::core::status::OfferedDeadlineMissedStatus** (p. 1871) status.

Implemented in **dds::domain::NoOpDomainParticipantListener** (p. 1847), and **dds::pub::NoOpPublisherListener** (p. 1853).

8.17.2.2 on_offered_incompatible_qos()

```
virtual void dds::pub::AnyDataWriterListener::on_offered_incompatible_qos (
             dds::pub::AnyDataWriter & writer,
            const  ::dds::core::status::OfferedIncompatibleQosStatus & status )  [pure virtual]
```

Handles the **dds::core::status::OfferedIncompatibleQosStatus** (p. 1873) status.

Implemented in **dds::domain::NoOpDomainParticipantListener** (p. 1847), and **dds::pub::NoOpPublisherListener** (p. 1854).

8.17.2.3 on_liveliness_lost()

```
virtual void dds::pub::AnyDataWriterListener::on_liveliness_lost (
             dds::pub::AnyDataWriter & writer,
            const  ::dds::core::status::LivelinessLostStatus & status )  [pure virtual]
```

Handles the **dds::core::status::LivelinessLostStatus** (p. 1655) status.

Implemented in **dds::domain::NoOpDomainParticipantListener** (p. 1847), and **dds::pub::NoOpPublisherListener** (p. 1854).

8.17.2.4 on_publication_matched()

```
virtual void dds::pub::AnyDataWriterListener::on_publication_matched (
             dds::pub::AnyDataWriter & writer,
            const  ::dds::core::status::PublicationMatchedStatus & status )  [pure virtual]
```

Handles the **dds::core::status::PublicationMatchedStatus** (p. 1990) status.

Implemented in **dds::domain::NoOpDomainParticipantListener** (p. 1848), and **dds::pub::NoOpPublisherListener** (p. 1854).

8.17.2.5 on_reliable_writer_cache_changed()

```
virtual void dds::pub::AnyDataWriterListener::on_reliable_writer_cache_changed (
             dds::pub::AnyDataWriter & writer,
            const  rti::core::status::ReliableWriterCacheChangedStatus & status )  [pure virtual]
```

<<***extension***>> *(p. 354)* Handles the dds::core::status::ReliableWriterCacheChangedStatus status

Implemented in **dds::domain::NoOpDomainParticipantListener** (p. 1848), and **dds::pub::NoOpPublisherListener** (p. 1854).

8.17.2.6 on_reliable_reader_activity_changed()

```
virtual void dds::pub::AnyDataWriterListener::on_reliable_reader_activity_changed (
             dds::pub::AnyDataWriter & writer,
            const  rti::core::status::ReliableReaderActivityChangedStatus & status )  [pure virtual]
```

<<***extension***>> *(p. 354)* Handles the dds::core::status::ReliableReaderActivityChangedStatus status

Implemented in **dds::domain::NoOpDomainParticipantListener** (p. 1848), and **dds::pub::NoOpPublisherListener** (p. 1855).

8.17.2.7 on_sample_removed()

```
virtual void dds::pub::AnyDataWriterListener::on_sample_removed (
             dds::pub::AnyDataWriter & writer,
            const  rti::core::Cookie & cookie )  [pure virtual]
```

<<***extension***>> *(p. 354)* Notifies when a sample is removed from **DataWriter** (p. 1138) queue.

Implemented in **dds::domain::NoOpDomainParticipantListener** (p. 1848), and **dds::pub::NoOpPublisherListener** (p. 1855).

8.17.2.8 on_instance_replaced()

```
virtual void dds::pub::AnyDataWriterListener::on_instance_replaced (
             dds::pub::AnyDataWriter & writer,
            const  dds::core::InstanceHandle & handle )  [pure virtual]
```

<<***extension***>> *(p. 354)* Notifies when an instance is replaced in **DataWriter** (p. 1138) queue.

Implemented in **dds::domain::NoOpDomainParticipantListener** (p. 1849), and **dds::pub::NoOpPublisherListener** (p. 1855).

8.17.2.9 on_application_acknowledgment()

```
virtual void dds::pub::AnyDataWriterListener::on_application_acknowledgment (
             dds::pub::AnyDataWriter & writer,
            const  rti::pub::AcknowledgmentInfo & acknowledgment_info )  [pure virtual]
```

<<***extension***>> *(p. 354)* Called when a sample is application-acknowledged.

Implemented in **dds::domain::NoOpDomainParticipantListener** (p. 1849), and **dds::pub::NoOpPublisherListener** (p. 1855).

8.17.2.10 on_service_request_accepted()

```
virtual void dds::pub::AnyDataWriterListener::on_service_request_accepted (
             dds::pub::AnyDataWriter & writer,
            const  rti::core::status::ServiceRequestAcceptedStatus & status )  [pure virtual]
```

<<***extension***>> *(p. 354)* Called when a sample that has been received on a the built-in service channel is intended for the **DataWriter** (p. 1138) that has installed this listener

Implemented in **dds::domain::NoOpDomainParticipantListener** (p. 1849), and **dds::pub::NoOpPublisherListener** (p. 1856).

8.18 dds::topic::AnyTopic Class Reference

<<***reference-type***>> *(p. 351)* This class provides an non-template holder for representing a **Topic** (p. 2474) of any type

`#include "dds/topic/AnyTopic.hpp"`

Public Member Functions

- template<typename T >
  AnyTopic (const **dds::topic::Topic**< T > &topic)
  *Create an **AnyTopic** (p. 825) that represents a generic, untyped **Topic** (p. 2474).*
- **dds::domain::DomainParticipant domain_participant** () const
  Gets the DomainParticipant this topic belongs to.
- **dds::core::status::InconsistentTopicStatus inconsistent_topic_status** ()
  Gets the InconsistentTopicStatus of this topic.
- const **dds::topic::qos::TopicQos qos** () const
  Get the TopicQos.
- void **qos** (const **dds::topic::qos::TopicQos** &the_qos)
  Sets the TopicQos.
- std::string **name** () const
  Gets the topic name.
- std::string **type_name** () const
  Gets the type name.
- void **close** ()
  *Closes the **Topic** (p. 2474).*
- template<typename T >
  Topic< T > **get** ()
  *Gets the typed **Topic** (p. 2474) from this **AnyTopic** (p. 825).*

Related Functions

(Note that these are not member functions.)

- template<typename T >
  Topic< T > **get** (const **AnyTopic** &any_topic)
  *Same as **AnyTopic::get()** (p. 827)*

8.18.1 Detailed Description

<<***reference-type***>> *(p. 351)* This class provides an non-template holder for representing a **Topic** (p. 2474) of any type

This class is useful for code that uses Topics of different types. It provides access to type-independent **Topic** (p. 2474) functions.

An **AnyTopic** (p. 825) is always created from a typed **Topic** (p. 2474), increasing its reference count. The member function **get()** (p. 827) allows getting the typed **Topic** (p. 2474) back.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 AnyTopic()

```
template<typename T >
dds::topic::AnyTopic::AnyTopic (
            const  dds::topic::Topic< T > & topic )  [inline]
```

Create an **AnyTopic** (p. 825) that represents a generic, untyped **Topic** (p. 2474).

Template Parameters

T	The type of the **Topic** (p. 2474) that this **AnyTopic** (p. 825) is being created from

Parameters

topic	The typed **Topic** (p. 2474) that this **AnyTopic** (p. 825) will represent

8.18.3 Member Function Documentation

8.18.3.1 domain_participant()

```
dds::domain::DomainParticipant dds::topic::AnyTopic::domain_participant ( ) const  [inline]
```

Gets the DomainParticipant this topic belongs to.

8.18.3.2 inconsistent_topic_status()

```
dds::core::status::InconsistentTopicStatus dds::topic::AnyTopic::inconsistent_topic_status ( )
[inline]
```

Gets the InconsistentTopicStatus of this topic.

8.18.3.3 qos() [1/2]

```
const  dds::topic::qos::TopicQos dds::topic::AnyTopic::qos ( ) const  [inline]
```

Get the TopicQos.

8.18.3.4 qos() [2/2]

```
void dds::topic::AnyTopic::qos (
            const  dds::topic::qos::TopicQos & the_qos )  [inline]
```

Sets the TopicQos.

8.18.3.5 name()

```
std::string dds::topic::AnyTopic::name ( ) const  [inline]
```

Gets the topic name.

8.18.3.6 type_name()

```
std::string dds::topic::AnyTopic::type_name ( ) const  [inline]
```

Gets the type name.

8.18.3.7 close()

```
void dds::topic::AnyTopic::close ( )  [inline]
```

Closes the **Topic** (p. 2474).

See also

Topic::close() (p. 1525)

8.18.3.8 get()

```
template<typename T >
Topic< T > dds::topic::AnyTopic::get ( )  [inline]
```

Gets the typed **Topic** (p. 2474) from this **AnyTopic** (p. 825).

Template Parameters

T	The type of the **Topic** (p. 2474) to extract

Returns

The typed **Topic** (p. 2474) that this **AnyTopic** (p. 825) represents

Exceptions

dds::core::InvalidDowncastError *(p. 1620)*	If this **AnyTopic** (p. 825) doesn't represent a Topic<T>.

Referenced by **get()**.

8.18.4 Friends And Related Function Documentation

8.18.4.1 get()

```
template<typename T >
Topic< T > get (
            const  AnyTopic & any_topic )  [related]
```

Same as **AnyTopic::get()** (p. 827)

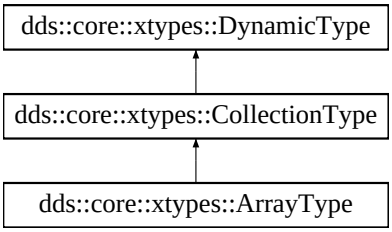
References **get()**.

8.19 dds::core::xtypes::ArrayType Class Reference

<<***value-type***>> *(p. 350)* Represents an IDL array type.

`#include <dds/core/xtypes/CollectionTypes.hpp>`

Inheritance diagram for dds::core::xtypes::ArrayType:



Public Member Functions

- **ArrayType** (const **dds::core::xtypes::DynamicType** &type, uint32_t the_dimension)

  Creates an unidimensional array.
- template<typename IntegerFwdIterator >
  ArrayType (const **dds::core::xtypes::DynamicType** &type, IntegerFwdIterator dimensions_begin, IntegerFwdIterator dimensions_end)

  Creates a multidimensional array.
- **ArrayType** (const **dds::core::xtypes::DynamicType** &type, std::initializer_list< uint32_t > dimensions)

  <<***C++11***>> *(*p. *353) Creates a multidimensional array*
- **ArrayType** (**dds::core::xtypes::DynamicType** &&type, uint32_t the_dimension)

  <<***C++11***>> *(*p. *353) Creates an unidimensional array*
- template<typename IntegerFwdIterator >
  ArrayType (**dds::core::xtypes::DynamicType** &&type, IntegerFwdIterator dimensions_begin, IntegerFwdIterator dimensions_end)

  <<***C++11***>> *(*p. *353) Creates a multidimensional array*
- **ArrayType** (**dds::core::xtypes::DynamicType** &&type, std::initializer_list< uint32_t > dimensions)

  <<***C++11***>> *(*p. *353) Creates a multidimensional array*
- uint32_t **dimension_count** () const

  Returns the number of dimensions.
- uint32_t **dimension** (uint32_t dimension_index) const

  Returns the ith dimension.
- uint32_t **total_element_count** () const

  Returns the sum of all the dimensions.

Additional Inherited Members

8.19.1 Detailed Description

<<***value-type***>> *(*p. *350)* Represents an IDL array type.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 ArrayType() [1/6]

```
dds::core::xtypes::ArrayType::ArrayType (
            const  dds::core::xtypes::DynamicType & type,
            uint32_t the_dimension )  [inline]
```

Creates an unidimensional array.

Parameters

type	The element type
the_dimension	The dimension of this unidimensional array

8.19.2.2 ArrayType() [2/6]

```
template<typename IntegerFwdIterator >
dds::core::xtypes::ArrayType::ArrayType (
            const  dds::core::xtypes::DynamicType & type,
            IntegerFwdIterator dimensions_begin,
            IntegerFwdIterator dimensions_end )  [inline]
```

Creates a multidimensional array.

For example, to create the following IDL array:
```
double x[10][20];
```

We can use this code:
```
std::vector<uint32_t> dim(2);
dim[0] = 10;
dim[1] = 20;
ArrayType my_array(primitive_type<double>(), dim.begin(), dim.end());
```

Parameters

type	The element type
dimensions_begin	The beginning of range of dimensions (uint32_t)
dimensions_end	The end of a range of dimensions (uint32_t)

8.19.2.3 ArrayType() [3/6]

```
dds::core::xtypes::ArrayType::ArrayType (
            const  dds::core::xtypes::DynamicType & type,
            std::initializer_list< uint32_t > dimensions )  [inline]
```

<<***C++11***>> *(p. 353)* Creates a multidimensional array

For example, to create the following IDL array:
```
double x[10][20];
```

We can use this code:
```
ArrayType my_array(primitive_type<double>(), {10, 20});
```

Parameters

type	The element type
dimensions	The dimensions of this array

8.19.2.4 ArrayType() [4/6]

```
dds::core::xtypes::ArrayType::ArrayType (
             dds::core::xtypes::DynamicType && type,
            uint32_t the_dimension )  [inline]
```

<<***C++11***>> *(p. 353)* Creates an unidimensional array

Parameters

type	The element type (to be moved)
the_dimension	The dimension of this unidimensional array

8.19.2.5 ArrayType() [5/6]

```
template<typename IntegerFwdIterator >
dds::core::xtypes::ArrayType::ArrayType (
             dds::core::xtypes::DynamicType && type,
            IntegerFwdIterator dimensions_begin,
            IntegerFwdIterator dimensions_end )  [inline]
```

<<***C++11***>> *(p. 353)* Creates a multidimensional array

Parameters

type	The element type (to be moved)
dimensions_begin	The beginning of range of dimensions (uint32_t)
dimensions_end	The end of a range of dimensions (uint32_t)

8.19.2.6 ArrayType() [6/6]

```
dds::core::xtypes::ArrayType::ArrayType (
             dds::core::xtypes::DynamicType && type,
            std::initializer_list< uint32_t > dimensions )  [inline]
```

<<***C++11***>> *(p. 353)* Creates a multidimensional array

Parameters

type	The element type (to be moved)
dimensions	The dimensions of this array

8.19.3 Member Function Documentation

8.19.3.1 dimension_count()

```
uint32_t dds::core::xtypes::ArrayType::dimension_count ( ) const
```

Returns the number of dimensions.

8.19.3.2 dimension()

```
uint32_t dds::core::xtypes::ArrayType::dimension (
            uint32_t dimension_index ) const
```

Returns the ith dimension.

8.19.3.3 total_element_count()

```
uint32_t dds::core::xtypes::ArrayType::total_element_count ( ) const
```

Returns the sum of all the dimensions.

8.20 rti::core::policy::AsynchronousPublisher Class Reference

<<***extension***>> *(p. 116)* Configures the mechanism to publish data using a separate thread

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **AsynchronousPublisher** ()

  Creates the default policy.
- **AsynchronousPublisher** & **disable_asynchronous_write** (bool the_disable_asynchronous_write)

  [DEPRECATED] *(*p. **296***)* *<<**extension**>>* *(*p. 116*) Disable asynchronous publishing.*
- bool **disable_asynchronous_write** () const

  Getter (see setter with the same name)
- **AsynchronousPublisher** & **thread** (const **rti::core::ThreadSettings** &the_thread)

  *<<**extension**>>* *(*p. 116*) Settings of the publishing thread.*
- const **rti::core::ThreadSettings** & **thread** () const

  Getter (see setter with the same name)
- **rti::core::ThreadSettings** & **thread** ()

  Getter (see setter with the same name)
- **AsynchronousPublisher** & **disable_asynchronous_batch** (bool the_disable_asynchronous_batch)

  [DEPRECATED] *(*p. **296***)* *<<**extension**>>* *(*p. 116*) Disable asynchronous batch flushing.*
- bool **disable_asynchronous_batch** () const

  Getter (see setter with the same name)
- **AsynchronousPublisher** & **asynchronous_batch_thread** (const **rti::core::ThreadSettings** &the_asynchronous_batch_thread)

  *<<**extension**>>* *(*p. 116*) Settings of the batch flushing thread.*
- const **rti::core::ThreadSettings** & **asynchronous_batch_thread** () const

  Getter (see setter with the same name)
- **rti::core::ThreadSettings** & **asynchronous_batch_thread** ()

  Getter (see setter with the same name)
- **AsynchronousPublisher** & **disable_topic_query_publication** (bool the_disable_topic_query_publication)

  [DEPRECATED] *(*p. **296***)* *<<**extension**>>* *(*p. 116*) Disable topic query publication.*
- bool **disable_topic_query_publication** () const

  Getter (see setter with the same name)
- **AsynchronousPublisher** & **topic_query_publication_thread** (const **rti::core::ThreadSettings** &the_topic_query_publication_thread)

  *<<**extension**>>* *(*p. 116*) Settings of the **rti::sub::TopicQuery*** *(*p. 2516*) publication thread.*
- const **rti::core::ThreadSettings** & **topic_query_publication_thread** () const

  Getter (see setter with the same name)
- **rti::core::ThreadSettings** & **topic_query_publication_thread** ()

  Getter (see setter with the same name)

Static Public Member Functions

- static **AsynchronousPublisher Enabled** (bool enable_async_batch=false)

  Returns a policy that enables asynchronous write and optionally asynchronous batch flushing.
- static **AsynchronousPublisher Disabled** ()

  Creates a policy that disables asynchronous writing.

8.20.1 Detailed Description

<<***extension***>> *(p. 116)* Configures the mechanism to publish data using a separate thread

Specifies the asynchronous publishing and asynchronous batch flushing settings of the **dds::pub::Publisher** (p. 1993) instances.

The QoS policy specifies whether asynchronous publishing and asynchronous batch flushing are enabled for the **dds::pub::DataWriter** (p. 1138) entities belonging to this **dds::pub::Publisher** (p. 1993). If so, the publisher will spawn up to two threads, one for asynchronous publishing and one for asynchronous batch flushing.

This policy also configures the settings of the **rti::sub::TopicQuery** (p. 2516) publication thread. The publisher will spawn this thread only if one or more DataWriters enable TopicQueries.

See also

rti::core::policy::Batch (p. 880).

rti::core::policy::PublishMode (p. 2013).

Entity:

dds::pub::Publisher (p. 1993)

Properties:

RxO (p. 514) = N/A

Changeable (p. 514) = **NO** (p. 514)

8.20.2 Usage

You can use this QoS policy to reduce the amount of time your application thread spends sending data.

The asynchronous *publisher* thread is shared by all **rti::core::policy::PublishModeKind_def::ASYNCHRONOUS** (p. 2019) **dds::pub::DataWriter** (p. 1138) instances that belong to this publisher and handles their data transmission chores.

The asynchronous *batch flushing* thread is shared by all **dds::pub::DataWriter** (p. 1138) instances with batching enabled that belong to this publisher.

This QoS policy also allows you to adjust the settings of the asynchronous publishing and the asynchronous batch flushing threads. To use different threads for two different **dds::pub::DataWriter** (p. 1138) entities, the instances must belong to different **dds::pub::Publisher** (p. 1993) instances.

A **dds::pub::Publisher** (p. 1993) must have asynchronous publishing enabled for its **dds::pub::DataWriter** (p. 1138) instances to write asynchronously.

A **dds::pub::Publisher** (p. 1993) must have asynchronous batch flushing enabled in order to flush the batches of its **dds::pub::DataWriter** (p. 1138) instances asynchronously. However, no asynchronous batch flushing thread will be started until the first **dds::pub::DataWriter** (p. 1138) instance with batching enabled is created from this **dds::pub::Publisher** (p. 1993).

8.20.3 Constructor & Destructor Documentation

8.20.3.1 AsynchronousPublisher()

```
rti::core::policy::AsynchronousPublisher::AsynchronousPublisher ( )  [inline]
```

Creates the default policy.

8.20.4 Member Function Documentation

8.20.4.1 disable_asynchronous_write() [1/2]

```
AsynchronousPublisher & rti::core::policy::AsynchronousPublisher::disable_asynchronous_write (
            bool the_disable_asynchronous_write )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(*p. *116)* Disable asynchronous publishing.

If set to true, any **dds::pub::DataWriter** (p. 1138) created with **rti::core::policy::PublishModeKind_def::ASYNCHRONOUS** (p. 2019) will fail with **dds::core::InconsistentPolicyError** (p. 1610).

[default] false

Categories

Immutable (p. 243), **Flow Control** (p. 188), **Concurrency** (p. 164)

Referenced by **rti::core::policy::Event::initial_count()**.

8.20.4.2 disable_asynchronous_write() [2/2]

```
bool rti::core::policy::AsynchronousPublisher::disable_asynchronous_write ( ) const
```

Getter (see setter with the same name)

8.20.4.3 thread() [1/3]

```
AsynchronousPublisher & rti::core::policy::AsynchronousPublisher::thread (
            const  rti::core::ThreadSettings & the_thread )
```

<<***extension***>> *(*p. *116)* Settings of the publishing thread.

There is only one asynchronous publishing thread per **dds::pub::Publisher** (p. 1993).

[default] priority below normal.
The actual value depends on your architecture:

For Windows: -2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] mask = **rti::core::ThreadSettingsKindMask::ThreadSettingsKindMask()** (p. 2455)

Categories

Immutable (p. 243), **Concurrency** (p. 164)

8.20.4.4 thread() [2/3]

```
const  rti::core::ThreadSettings & rti::core::policy::AsynchronousPublisher::thread ( ) const
```

Getter (see setter with the same name)

8.20.4.5 thread() [3/3]

```
rti::core::ThreadSettings & rti::core::policy::AsynchronousPublisher::thread ( )
```

Getter (see setter with the same name)

8.20.4.6 disable_asynchronous_batch() [1/2]

```
AsynchronousPublisher & rti::core::policy::AsynchronousPublisher::disable_asynchronous_batch (
            bool the_disable_asynchronous_batch )
```

[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Disable asynchronous batch flushing.

If set to true, any **dds::pub::DataWriter** (p. 1138) created with batching enabled will fail with **dds::core::Inconsistent↩ PolicyError** (p. 1610).

If **rti::core::policy::Batch::max_flush_delay** (p. 884) is different than **dds::core::Duration::infinite()** (p. 1451), **rti↩ ::core::policy::AsynchronousPublisher::disable_asynchronous_batch** (p. 836) must be set false.

[default] false

Categories

Immutable (p. 243), **Concurrency** (p. 164), **Batching** (p. 161)

Referenced by **rti::core::policy::Event::initial_count()**.

8.20.4.7 disable_asynchronous_batch() [2/2]

```
bool rti::core::policy::AsynchronousPublisher::disable_asynchronous_batch ( ) const
```

Getter (see setter with the same name)

8.20.4.8 asynchronous_batch_thread() [1/3]

```
AsynchronousPublisher & rti::core::policy::AsynchronousPublisher::asynchronous_batch_thread (
            const  rti::core::ThreadSettings & the_asynchronous_batch_thread )
```

<<***extension***>> *(*p. *116)* Settings of the batch flushing thread.

There is only one asynchronous batch flushing thread per **dds::pub::Publisher** (p. 1993).

[default] priority below normal.
The actual value depends on your architecture:

For Windows: -2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`. **[default]** The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] mask = **rti::core::ThreadSettingsKindMask::ThreadSettingsKindMask()** (p. 2455)

Categories

Immutable (p. 243), **Concurrency** (p. 164), **Batching** (p. 161)

8.20.4.9 asynchronous_batch_thread() [2/3]

```
const rti::core::ThreadSettings & rti::core::policy::AsynchronousPublisher::asynchronous_batch_↩
thread ( ) const
```

Getter (see setter with the same name)

8.20.4.10 asynchronous_batch_thread() [3/3]

```
rti::core::ThreadSettings & rti::core::policy::AsynchronousPublisher::asynchronous_batch_thread (
)
```

Getter (see setter with the same name)

8.20.4.11 disable_topic_query_publication() [1/2]

```
AsynchronousPublisher & rti::core::policy::AsynchronousPublisher::disable_topic_query_publication
(
            bool the_disable_topic_query_publication )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Disable topic query publication.

If set to true, any **dds::pub::DataWriter** (p. 1138) created with **rti::core::policy::TopicQueryDispatch::enable** (p. 2524) set to true will fail with **dds::core::InconsistentPolicyError** (p. 1610).

[default] false

Categories

Immutable (p. 243), **Concurrency** (p. 164), **Topic Queries** (p. 239)

8.20.4.12 disable_topic_query_publication() [2/2]

```
bool rti::core::policy::AsynchronousPublisher::disable_topic_query_publication ( ) const
```

Getter (see setter with the same name)

8.20.4.13 topic_query_publication_thread() [1/3]

```
AsynchronousPublisher & rti::core::policy::AsynchronousPublisher::topic_query_publication_thread
(
            const  rti::core::ThreadSettings & the_topic_query_publication_thread )
```

<<***extension***>> *(*p. *116)* Settings of the **rti::sub::TopicQuery** (p. 2516) publication thread.

There is only one TopicQuery publication thread per **dds::pub::Publisher** (p. 1993). This thread will exist as long as one or more **dds::pub::DataWriter** (p. 1138) enables TopicQueries (via **rti::core::policy::TopicQueryDispatch** (p. 2522)).

[default] priority below normal.
The actual value depends on your architecture:

For Windows: -2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`. **[default]** The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] mask = **rti::core::ThreadSettingsKindMask::ThreadSettingsKindMask()** (p. 2455)

Categories

Immutable (p. 243), **Concurrency** (p. 164), **Topic Queries** (p. 239)

8.20.4.14 topic_query_publication_thread() [2/3]

```
const  rti::core::ThreadSettings & rti::core::policy::AsynchronousPublisher::topic_query_publication↩
_thread ( ) const
```

Getter (see setter with the same name)

8.20.4.15 topic_query_publication_thread() [3/3]

```
rti::core::ThreadSettings & rti::core::policy::AsynchronousPublisher::topic_query_publication_↩
thread ( )
```

Getter (see setter with the same name)

8.20.4.16 Enabled()

```
static AsynchronousPublisher rti::core::policy::AsynchronousPublisher::Enabled (
            bool enable_async_batch = false ) [inline], [static]
```

Returns a policy that enables asynchronous write and optionally asynchronous batch flushing.

It uses the default thread settings.

For example:

```
using namespace rti::core::policy;
publisher_qos « AsynchronousPublisher::Enabled()
```

8.20.4.17 Disabled()

```
static AsynchronousPublisher rti::core::policy::AsynchronousPublisher::Disabled ( ) [inline],
[static]
```

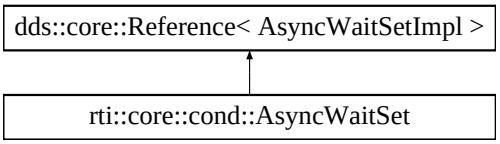
Creates a policy that disables asynchronous writing.

8.21 rti::core::cond::AsyncWaitSet Class Reference

<<***extension***>> *(p. 354)* Dispatches **dds::core::cond::Condition** (p. 948) objects using separate threads of execution. Unlike a normal **dds::core::cond::WaitSet** (p. 2617), which operates on the current thread, an **AsyncWaitSet** (p. 840) uses a thread pool.

```
#include <rti/core/cond/AsyncWaitSet.hpp>
```

Inheritance diagram for rti::core::cond::AsyncWaitSet:



Public Member Functions

- **AsyncWaitSet** ()

  *Constructor without arguments that create an **rti::core::cond::AsyncWaitSet** (p. 840) with default property.*
- **AsyncWaitSet** (const **AsyncWaitSetProperty** &the_property)

  *Single-argument constructor that allows creating a a **rti::core::cond::AsyncWaitSet** (p. 840) with custom behavior.*
- **AsyncWaitSet** (const **AsyncWaitSetProperty** &the_property, **AsyncWaitSetListener** *listener)

  *Constructor that allows specifying a **rti::core::cond::AsyncWaitSetListener** (p. 856).*
- void **start** ()

  *Initiates the asynchronous wait on this **rti::core::cond::AsyncWaitSet** (p. 840).*
- void **start** (**AsyncWaitSetCompletionToken** completion_token)

  *Initiates the asynchronous wait on this **rti::core::cond::AsyncWaitSet** (p. 840).*
- void **stop** ()

  *Initiates the stop procedure on this **rti::core::cond::AsyncWaitSet** (p. 840) that will stop the asynchronous wait.*
- void **stop** (**AsyncWaitSetCompletionToken** completion_token)

  *Initiates the stop procedure on this **rti::core::cond::AsyncWaitSet** (p. 840) that will stop the asynchronous wait.*
- **AsyncWaitSet** & **attach_condition** (**dds::core::cond::Condition** condition)

  *Attaches the specified **dds::core::cond::Condition** (p. 948) to this **rti::core::cond::AsyncWaitSet** (p. 840).*
- **AsyncWaitSet** & **attach_condition** (**dds::core::cond::Condition** condition, **AsyncWaitSetCompletion↩ Token** completion_token)

  *Attaches the specified **dds::core::cond::Condition** (p. 948) to this **rti::core::cond::AsyncWaitSet** (p. 840).*
- **AsyncWaitSet** & **detach_condition** (**dds::core::cond::Condition** condition)

  *Deaches the specified **dds::core::cond::Condition** (p. 948) from this **rti::core::cond::AsyncWaitSet** (p. 840).*
- **AsyncWaitSet** & **detach_condition** (**dds::core::cond::Condition** condition, **AsyncWaitSetCompletion↩ Token** completion_token)

  *Detaches the specified **dds::core::cond::Condition** (p. 948) from this **rti::core::cond::AsyncWaitSet** (p. 840).*
- **AsyncWaitSet** & **operator+=** (**dds::core::cond::Condition** condition)

  *Attaches the specified **dds::core::cond::Condition** (p. 948) to this **rti::core::cond::AsyncWaitSet** (p. 840).*
- **AsyncWaitSet** & **operator-=** (**dds::core::cond::Condition** condition)

  *Deaches the specified **dds::core::cond::Condition** (p. 948) from this **rti::core::cond::AsyncWaitSet** (p. 840).*
- void **unlock_condition** (**dds::core::cond::Condition** condition)

  *Allows the **dds::core::cond::Condition** (p. 948) under dispatch to be available for concurrent dispatch from another thread from the pool.*
- **AsyncWaitSetProperty** **property** ()

  *Retrieves the **rti::core::cond::AsyncWaitSetProperty** (p. 858) configuration of the associated **rti::core::cond::Async↩ WaitSet** (p. 840).*
- ConditionSeq **conditions** () const

  *Returns the list of attached **dds::core::cond::Condition** (p. 948) (s).*
- ConditionSeq & **conditions** (ConditionSeq &attached_conditions) const

  *Retrieves the list of attached **dds::core::cond::Condition** (p. 948) (s).*

8.21.1 Detailed Description

<<***extension***>> *(p. 354)* Dispatches **dds::core::cond::Condition** (p. 948) objects using separate threads of execution. Unlike a normal **dds::core::cond::WaitSet** (p. 2617), which operates on the current thread, an **AsyncWaitSet** (p. 840) uses a thread pool.

rti::core::cond::AsyncWaitSet (p. 840) provides a proactive model to process application events through **dds::core↩ ::cond::Condition** (p. 948) objects. **rti::core::cond::AsyncWaitSet** (p. 840) owns a pool of threads to asynchronously wait for the attached **dds::core::cond::Condition** (p. 948) objects to trigger and dispatch them upon wakeup. The asynchronous behavior is the main key different with regards to the **dds::core::cond::WaitSet** (p. 2617).

The class diagram and its collaborators is shown below:

8.21.2 AsyncWaitSet Thread Orchestration

rti::core::cond::AsyncWaitSet (p. 840) internally applies a leader-follower pattern for the orchestration of the thread pool. Once a **rti::core::cond::AsyncWaitSet** (p. 840) starts, it will create the thread pool of `M` threads from which only one thread will become the Leader thread, and remaining threads will become the Followers, where:

- The `Leader` thread is the one waiting for the attached **dds::core::cond::Condition** (p. 948) to trigger. Remaining threads in the pool, if any, are either idle awaiting to become the leader or busy while processing active **dds::core::cond::Condition** (p. 948).
- Upon wait wakeup, the `Leader` thread resigns its leader status to become a `Processor` thread and dispatch the next active **dds::core::cond::Condition** (p. 948) through the **dds::core::cond::Condition::dispatch** (p. 949) operation.
- One of the `Follower` threads wakes up and becomes the new leader to resume the wait for **dds::core::cond↩ ::Condition** (p. 948).

"Thread orchestration in a rti::core::cond::AsyncWaitSet"

This behavior implies the following considerations:

- Only one thread a time can wait for the attached **dds::core::cond::Condition** (p. 948) objects to trigger. From a pool of `M` threads, only one is the leader, `P` are processing active **dds::core::cond::Condition** (p. 948), and `F` are idle followers.
- A thread can dispatch only one active **dds::core::cond::Condition** (p. 948) at a time.
- **rti::core::cond::AsyncWaitSet** (p. 840) efficiently distributes threads to dispatch **dds::core::cond::Condition** (p. 948) objects on demand. This avoids underutilizing a thread if **dds::core::cond::Condition** (p. 948) objects do not trigger or trigger unfrequently.
- At a given time, all the threads in the pool could be in processing state. In this situation, the **rti::core::cond::↩ AsyncWaitSet** (p. 840) is not able to wait for more **dds::core::cond::Condition** (p. 948) objects until one thread becomes the leader.

rti::core::cond::AsyncWaitSet (p. 840) has a built-in dispatcher that guarantees fairness and avoids starvation of **dds::core::cond::Condition** (p. 948) objects. By applying a round-robin distribution policy, each attached and active **dds::core::cond::Condition** (p. 948) is dispatched within a finite period of time, assuming the **functor handler** (p. 2134) always return control after the **dds::core::cond::Condition::dispatch** (p. 949) operation.

8.21.3 AsyncWaitSet Thread Safety

A key aspect of the **rti::core::cond::AsyncWaitSet** (p. 840) is the thread safety. **rti::core::cond::AsyncWaitSet** (p. 840) interface is thread safe, so you can concurrently call any operation on the **rti::core::cond::AsyncWaitSet** (p. 840) object from multiple threads in your application.

Furthermore, **rti::core::cond::AsyncWaitSet** (p. 840) also safely interacts with its own thread pool. Internally, the **rti::core::cond::AsyncWaitSet** (p. 840) applies the asynchronous completion token pattern to perform activities that involve synchronization with the thread pool.

For instance to detach a **dds::core::cond::Condition** (p. 948), the **rti::core::cond::AsyncWaitSet** (p. 840) generates an internal request to its thread pool to process it. As soon as the detachment completes, the thread pool provides the notification through an associated completion token.

Note

The asynchronous completion token behavior only takes place if the **rti::core::cond::AsyncWaitSet** (p. 840) is started. Otherwise the internal request will be directly executed by the calling thread.

For a finer control on this behavior, each **rti::core::cond::AsyncWaitSet** (p. 840) operation where this applies comes in two flavors:

- `Default:` the operation hides all the details of the completion token and returns after the operation completes. Operations of this kind internally use an implicit **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853). The **rti::core::cond::AsyncWaitSet** (p. 840) creates and reuses **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853) objects as needed. This is the recommended flavor unless your application has special resource needs.
- `With completion token:` An overloaded version of the default one that also receives an **rti::core::cond↩ ::AsyncWaitSetCompletionToken** (p. 853) object on which you can wait on at any time for the actual operation to complete. This flavor is available to assist applications with resource constraints and that want more control on the interaction with the thread pool of the **rti::core::cond::AsyncWaitSet** (p. 840).

8.21.3.1 Condition Locking

rti::core::cond::AsyncWaitSet (p. 840) incorporates a safety mechanism that prevents calling **dds::core::cond::↩ Condition::dispatch** (p. 949) concurrently. **rti::core::cond::AsyncWaitSet** (p. 840) locks the **dds::core::cond::↩ Condition** (p. 948) while a processor thread is dispatching it so no other thread within the pool can dispatch it again.

This mechanism ensures not only unexpected concurrent dispatch of a **dds::core::cond::Condition** (p. 948) but also spurious thread activity. Because it is responsibility of your application to reset the Condition trigger, there is a period of time in which the dispatched condition may remain active, causing the **rti::core::cond::AsyncWaitSet** (p. 840) to enter in a continous immediate wakeup from the wait. This behavior typically leads to thread hogging and high CPU usage.

Nevertheless, your application may still want to receive concurrent and controlled dispatch notifications. **rti::core↩ ::cond::AsyncWaitSet** (p. 840) will still allows you to unlock a **dds::core::cond::Condition** (p. 948) so any other available thread can dispatch the same condition concurrently while preventing the above mentioned problems. You can achive this by calling **rti::core::cond::AsyncWaitSet::unlock_condition** (p. 852) on the Condition being dispatched within the dispatch callback. Note that the **AsyncWaitSet** (p. 840) locks a Condition each time it dispatches it. Hence you need to unlock the Condition each time you want to enable a concurrent dispatch.

8.21.4 AsyncWaitSet Events and Resources

Besides **dds::core::cond::Condition** (p. 948) processing, you can listen to other kind of internal events related to the **rti::core::cond::AsyncWaitSet** (p. 840) and its thread pool by means of the **rti::core::cond::AsyncWaitSetListener** (p. 856).

rti::core::cond::AsyncWaitSet (p. 840) exposes operations to start and stop the asynchronous wait, which involves the creation and deletion of the thread pool respectively.

rti::core::cond::AsyncWaitSet (p. 840) relies on thread-specific storage to provide the described functionality. Each application thread that calls an operation on a **rti::core::cond::AsyncWaitSet** (p. 840) will generate resources that will be associated with such thread. You can free these resources upon thread termination by calling DomainParticipant↩ Factory::unregister_thread.

MT Safety:

Safe.

See also

dds::core::cond::WaitSet (p. 2617)

dds::core::cond::Condition (p. 948)

rti::core::cond::AsyncWaitSetListener (p. 856)

rti::core::cond::AsyncWaitSetCompletionToken (p. 853).

rti::core::cond::AsyncWaitSetProperty (p. 858)

8.21.5 Constructor & Destructor Documentation

8.21.5.1 AsyncWaitSet() [1/3]

```
rti::core::cond::AsyncWaitSet::AsyncWaitSet ( )  [inline]
```

Constructor without arguments that create an **rti::core::cond::AsyncWaitSet** (p. 840) with default property.

See also

rti::core::cond::AsyncWaitSetProperty::AsyncWaitSetProperty() (p. 859)

8.21.5.2 AsyncWaitSet() [2/3]

```
rti::core::cond::AsyncWaitSet::AsyncWaitSet (
            const  AsyncWaitSetProperty & the_property )  [inline]
```

Single-argument constructor that allows creating a a **rti::core::cond::AsyncWaitSet** (p. 840) with custom behavior.

You can provide **rti::core::cond::AsyncWaitSetProperty::AsyncWaitSetProperty()** (p. 859) as `property` to create an **rti::core::cond::AsyncWaitSet** (p. 840) with default behavior.

The **rti::core::cond::AsyncWaitSet** (p. 840) is created with no listener installed.

Parameters

the_property	<<***in***>> *(p. 355)* configuration **rti::core::cond::AsyncWaitSetProperty** (p. 858)

8.21.5.3 AsyncWaitSet() [3/3]

```
rti::core::cond::AsyncWaitSet::AsyncWaitSet (
            const  AsyncWaitSetProperty & the_property,
             AsyncWaitSetListener * listener )  [inline]
```

Constructor that allows specifying a **rti::core::cond::AsyncWaitSetListener** (p. 856).

Creates a new **rti::core::cond::AsyncWaitSet** (p. 840) with the specified property **rti::core::cond::AsyncWaitSet↩ Property** (p. 858) and **rti::core::cond::AsyncWaitSetListener** (p. 856).

Parameters

the_property	<<***in***>> *(p. 355)* configuration **rti::core::cond::AsyncWaitSetProperty** (p. 858)
listener	<<***in***>> *(p. 355)* the **rti::core::cond::AsyncWaitSetListener** (p. 856). Cannot be NULL.

8.21.6 Member Function Documentation

8.21.6.1 start() [1/2]

```
void rti::core::cond::AsyncWaitSet::start ( )  [inline]
```

Initiates the asynchronous wait on this **rti::core::cond::AsyncWaitSet** (p. 840).

This operation blocks until the start request completes. Upon successful return, it is guaranteed that this **rti::core↩ ::cond::AsyncWaitSet** (p. 840) has initiated the asynchronous wait and dispatch.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::core::cond::AsyncWaitSet::start(AsyncWaitSetCompletionToken) (p. 845)

rti::core::cond::AsyncWaitSet::stop (p. 846)

8.21.6.2 start() [2/2]

```
void rti::core::cond::AsyncWaitSet::start (
            AsyncWaitSetCompletionToken completion_token )  [inline]
```

Initiates the asynchronous wait on this **rti::core::cond::AsyncWaitSet** (p. 840).

If this operation succeeds, a start request has been scheduled and your application can use the provided `completion_token` to wait for this **rti::core::cond::AsyncWaitSet** (p. 840) to process the request. If the **rti::core::cond::AsyncWaitSetCompletionToken::wait** (p. 855) operation returns successfully, it is guranteed that the thread pool has been created and the leader thread is waiting for the attached **dds::core::cond::Condition** (p. 948) to trigger.

Once this **rti::core::cond::AsyncWaitSet** (p. 840) is started, attached **dds::core::cond::Condition** (p. 948) will be dispatched through the **dds::core::cond::Condition::dispatch** (p. 949) operation when they trigger.

The start procedure causes the **rti::core::cond::AsyncWaitSet** (p. 840) to spawn all the threads within the thread pool, which involves the underlying operating system to allocate the associated thread stack and context for each thread. If a **rti::core::cond::AsyncWaitSetListener** (p. 856) is installed, this **rti::core::cond::AsyncWaitSet** (p. 840) will sequentially invoke the **rti::core::cond::AsyncWaitSetListener::on_thread_spawned** (p. 857) once per spwaned thread.

A **rti::core::cond::AsyncWaitSet** (p. 840) can be restarted after a stop. If this **rti::core::cond::AsyncWaitSet** (p. 840) is already started, this operation will return immediately with success, and waiting on the `completion_token` will also return immediately with success.

Parameters

completion_token	<<***inout***>> *(p. 355)* a valid **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853) instance that can be used by your application to wait for the start request to complete. You can provide the special sentinel **rti::core::cond::AsyncWaitSetCompletionToken::Ignore()** (p. 508).

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::core::cond::AsyncWaitSet::start (p. 845)

rti::core::cond::AsyncWaitSet::start(AsyncWaitSetCompletionToken) (p. 845)

dds::core::cond::WaitSet::wait (p. 2621)

8.21.6.3 stop() [1/2]

```
void rti::core::cond::AsyncWaitSet::stop ( )  [inline]
```

Initiates the stop procedure on this **rti::core::cond::AsyncWaitSet** (p. 840) that will stop the asynchronous wait.

This operation will block until the stop request completes. Upon successful return, it is guaranteed that this **rti::core::cond::AsyncWaitSet** (p. 840) stopped the asynchronous wait and dispatch.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::core::cond::AsyncWaitSet::start(AsyncWaitSetCompletionToken) (p. 845)

rti::core::cond::AsyncWaitSet::stop (p. 846)

8.21.6.4 stop() [2/2]

```
void rti::core::cond::AsyncWaitSet::stop (
            AsyncWaitSetCompletionToken completion_token )  [inline]
```

Initiates the stop procedure on this **rti::core::cond::AsyncWaitSet** (p. 840) that will stop the asynchronous wait.

If this operation succeeds, a stop request has been scheduled and your application can use the provided `completion_token` to wait for this **rti::core::cond::AsyncWaitSet** (p. 840) to process the request. If the **rti::core::cond::AsyncWaitSetCompletionToken::wait** (p. 855) operation returns successfully, it is guranteed that the thread pool has been deleted and this **rti::core::cond::AsyncWaitSet** (p. 840) no longer process any of the attached **dds::core::cond::Condition** (p. 948) objects.

Once this **rti::core::cond::AsyncWaitSet** (p. 840) is stopped, the **dds::core::cond::Condition::dispatch** (p. 949) will no longer be called on any of the attached **dds::core::cond::Condition** (p. 948), no matter what their trigger value is.

The stop procedure causes the **rti::core::cond::AsyncWaitSet** (p. 840) to delete all the threads within the thread pool, which involves the underlying operating system to release the associated thread stack and context of each thread. If a **rti::core::cond::AsyncWaitSetListener** (p. 856) is installed, this **rti::core::cond::AsyncWaitSet** (p. 840) will sequentially invoke the **rti::core::cond::AsyncWaitSetListener::on_thread_deleted** (p. 857) once per deleted thread.

If this **rti::core::cond::AsyncWaitSet** (p. 840) is already stopped, this operation will return immediately with success, and waiting on the `completion_token` will also return immediately with success.

Parameters

completion_token	<<***inout***>> *(p. 355)* a valid **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853) instance that can be used by your application to wait for the stop request to complete. You can provide the special sentinel **rti::core::cond::AsyncWaitSetCompletionToken::Ignore()** (p. 508).

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::core::cond::AsyncWaitSet::stop (p. 846)

rti::core::cond::AsyncWaitSet::start(AsyncWaitSetCompletionToken) (p. 845)

8.21.6.5 attach_condition() [1/2]

```
AsyncWaitSet & rti::core::cond::AsyncWaitSet::attach_condition (
            dds::core::cond::Condition condition ) [inline]
```

Attaches the specified **dds::core::cond::Condition** (p. 948) to this **rti::core::cond::AsyncWaitSet** (p. 840).

This operation will block until the attach request completes. Upon successful return, it is guaranteed that the specified **dds::core::cond::Condition** (p. 948) is attached.

If the same **dds::core::cond::Condition** (p. 948) is attached multiple times, only the first attachment has any effect. Subsequent attempts to attach the same **dds::core::cond::Condition** (p. 948) are silently ignored. As a result, the **dds::core::cond::Condition** (p. 948) is only considered attached once and must only be detached once.

Parameters

condition	<<***in***>> *(p. 355)* **dds::core::cond::Condition** (p. 948) to be attached.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::core::cond::AsyncWaitSet::attach_condition(dds::core::cond::Condition, AsyncWaitSetCompletion↩ Token) (p. 848)

8.21.6.6 attach_condition() [2/2]

```
AsyncWaitSet & rti::core::cond::AsyncWaitSet::attach_condition (
            dds::core::cond::Condition condition,
            AsyncWaitSetCompletionToken completion_token ) [inline]
```

Attaches the specified **dds::core::cond::Condition** (p. 948) to this **rti::core::cond::AsyncWaitSet** (p. 840).

If this operation succeeds, an attach request has been scheduled and your application can use the output parameter `completion_token` to wait for this **rti::core::cond::AsyncWaitSet** (p. 840) to process the request. **rti::core::cond::AsyncWaitSetCompletionToken::wait** (p. 855) operation returns successfully, it is guaranteed that the **dds::core::cond::Condition** (p. 948) is attached to this **rti::core::cond::AsyncWaitSet** (p. 840).

Once the **dds::core::cond::Condition** (p. 948) is attached, its trigger value may cause the leader thread of the **rti::core::cond::AsyncWaitSet** (p. 840) to wake up call the **dds::core::cond::Condition::dispatch** (p. 949) operation.

dds::core::cond::Condition (p. 948) may be attached at any time independently of the state of the **rti::core::cond::AsyncWaitSet** (p. 840).

Parameters

condition	*<<**in**>> (p. 355)* **dds::core::cond::Condition** (p. 948) to be attached.
completion_token	*<<**inout**>> (p. 355)* a valid **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853) instance that can be used by your application to wait for the attach request to complete. You can provide the special sentinel **rti::core::cond::AsyncWaitSetCompletionToken::Ignore()** (p. 508).

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::core::cond::AsyncWaitSet::attach_condition (p. 848)

rti::core::cond::AsyncWaitSet::detach_condition(dds::core::cond::Condition, AsyncWaitSetCompletionToken) (p. 850)

rti::core::cond::AsyncWaitSetCompletionToken::wait (p. 855)

8.21.6.7 detach_condition() [1/2]

```
AsyncWaitSet & rti::core::cond::AsyncWaitSet::detach_condition (
            dds::core::cond::Condition condition ) [inline]
```

Deaches the specified **dds::core::cond::Condition** (p. 948) from this **rti::core::cond::AsyncWaitSet** (p. 840).

This operation blocks until the detach request completes. Upon successful return, it is guaranteed that the specified **dds::core::cond::Condition** (p. 948) is detached.

Parameters

condition	*<<**in**>> (p. 355)* **dds::core::cond::Condition** (p. 948) to be detached.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::core::cond::AsyncWaitSet::detach_condition(dds::core::cond::Condition, AsyncWaitSetCompletion↩ Token) (p. 850)

8.21.6.8 detach_condition() [2/2]

```
AsyncWaitSet & rti::core::cond::AsyncWaitSet::detach_condition (
            dds::core::cond::Condition condition,
            AsyncWaitSetCompletionToken completion_token )  [inline]
```

Detaches the specified **dds::core::cond::Condition** (p. 948) from this **rti::core::cond::AsyncWaitSet** (p. 840).

If this operation succeeds, a detach request has been scheduled and your application can use the provided `completion_token` to wait for this **rti::core::cond::AsyncWaitSet** (p. 840) to process the request. If the **rti↩ ::core::cond::AsyncWaitSetCompletionToken::wait** (p. 855) operation returns successfully, it is guaranteed that the **dds::core::cond::Condition** (p. 948) is detached from this **rti::core::cond::AsyncWaitSet** (p. 840).

Once the **dds::core::cond::Condition** (p. 948) is detached, it is guaranteed that the **rti::core::cond::AsyncWaitSet** (p. 840) will no longer process it so it is safe for your application to release any resources associated with the detached **dds::core::cond::Condition** (p. 948).

dds::core::cond::Condition (p. 948) may be detached at any time independently of the state of the **rti::core::cond↩ ::AsyncWaitSet** (p. 840).

If the same **dds::core::cond::Condition** (p. 948) had been previously attached multiple times, only a single detach is required; the **dds::core::cond::Condition** (p. 948) is considered attached only once.

Parameters

condition	*<<**in**>> (p. 355)* **dds::core::cond::Condition** (p. 948) to be detached.
completion_token	*<<**inout**>> (p. 355)* a valid **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853) instance that can be used by your application to wait for the detach request to complete. You can provide the special sentinel **rti::core::cond::AsyncWaitSetCompletionToken::Ignore()** (p. 508).

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::core::cond::AsyncWaitSet::detach_condition (p. 849)

rti::core::cond::AsyncWaitSet::attach_condition(dds::core::cond::Condition, AsyncWaitSetCompletion↩ Token) (p. 848)

rti::core::cond::AsyncWaitSetCompletionToken::wait (p. 855)

8.21.6.9 operator+=()

```
AsyncWaitSet & rti::core::cond::AsyncWaitSet::operator+= (
            dds::core::cond::Condition condition ) [inline]
```

Attaches the specified **dds::core::cond::Condition** (p. 948) to this **rti::core::cond::AsyncWaitSet** (p. 840).

This operation will block until the attach request completes. Upon successful return, it is guaranteed that the specified **dds::core::cond::Condition** (p. 948) is attached.

If the same **dds::core::cond::Condition** (p. 948) is attached multiple times, only the first attachment has any effect. Subsequent attempts to attach the same **dds::core::cond::Condition** (p. 948) are silently ignored. As a result, the **dds::core::cond::Condition** (p. 948) is only considered attached once and must only be detached once.

Parameters

condition	<<***in***>> *(p. 355)* **dds::core::cond::Condition** (p. 948) to be attached.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::core::cond::AsyncWaitSet::attach_condition(dds::core::cond::Condition, AsyncWaitSetCompletion↩ Token) (p. 848)

8.21.6.10 operator-=()

```
AsyncWaitSet & rti::core::cond::AsyncWaitSet::operator-= (
            dds::core::cond::Condition condition ) [inline]
```

Deaches the specified **dds::core::cond::Condition** (p. 948) from this **rti::core::cond::AsyncWaitSet** (p. 840).

This operation blocks until the detach request completes. Upon successful return, it is guaranteed that the specified **dds::core::cond::Condition** (p. 948) is detached.

Parameters

condition	<<***in***>> *(*p. *355)* **dds::core::cond::Condition** (p. 948) to be detached.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::core::cond::AsyncWaitSet::detach_condition(dds::core::cond::Condition, AsyncWaitSetCompletion↩ Token) (p. 850)

8.21.6.11 unlock_condition()

```
void rti::core::cond::AsyncWaitSet::unlock_condition (
            dds::core::cond::Condition condition )  [inline]
```

Allows the **dds::core::cond::Condition** (p. 948) under dispatch to be available for concurrent dispatch from another thread from the pool.

This operation can be called from the dispatch callback of the **dds::core::cond::Condition** (p. 948) this **rti::core↩ ::cond::AsyncWaitSet** (p. 840) is dispatching. After succesfully calling this operation, if the **dds::core::cond::↩ Condition** (p. 948) becomes active this **rti::core::cond::AsyncWaitSet** (p. 840) is allowed to dispatch it again from any available thread from the pool.

You may call this operation any time you need the same **dds::core::cond::Condition** (p. 948) to be dispatched concurrently.

This operation will fail with **dds::core::PreconditionNotMetError** (p. 1941) if you call it from a different context than the dispatch callback or on a different **dds::core::cond::Condition** (p. 948).

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.21.6.12 property()

```
AsyncWaitSetProperty rti::core::cond::AsyncWaitSet::property ( )  [inline]
```

Retrieves the **rti::core::cond::AsyncWaitSetProperty** (p. 858) configuration of the associated **rti::core::cond::↩ AsyncWaitSet** (p. 840).

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.21.6.13 conditions() [1/2]

```
ConditionSeq rti::core::cond::AsyncWaitSet::conditions ( ) const  [inline]
```

Returns the list of attached **dds::core::cond::Condition** (p. 948) (s).

8.21.6.14 conditions() [2/2]

```
ConditionSeq & rti::core::cond::AsyncWaitSet::conditions (
            ConditionSeq & attached_conditions ) const  [inline]
```

Retrieves the list of attached **dds::core::cond::Condition** (p. 948) (s).

Parameters

attached_conditions	<<***inout***>> *(p. 355)* a ConditionSeq object where the list of attached conditions will be returned.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::core::cond::AsyncWaitSet::attach_condition (p. 848)

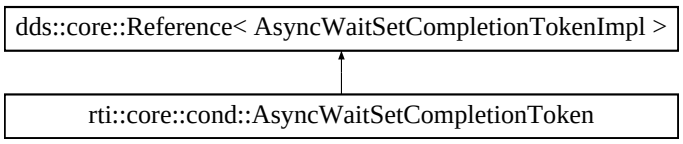
rti::core::cond::AsyncWaitSet::detach_condition (p. 849)

8.22 rti::core::cond::AsyncWaitSetCompletionToken Class Reference

Implementation of the completion token role element of the asynchronous completion token pattern that is part of the **AsyncWaitSet** (p. 840) behavior.

```
#include <AsyncWaitSet.hpp>
```

Inheritance diagram for rti::core::cond::AsyncWaitSetCompletionToken:



Public Member Functions

- **AsyncWaitSetCompletionToken** (**AsyncWaitSet** &aws)

  *Creates a new **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853).*
- void **wait** (const **dds::core::Duration** &max_wait= **dds::core::Duration::infinite**())

  *Waits for the request associated to an operation made on the **rti::core::cond::AsyncWaitSet** (p. 840) to complete.*

Static Public Member Functions

- static **AsyncWaitSetCompletionToken Ignore** ()

  *For the operations that allow an **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853), this sentinel can be provided to indicate an **rti::core::cond::AsyncWaitSet** (p. 840) to perform the action associating a 'null' completion token.*

8.22.1 Detailed Description

Implementation of the completion token role element of the asynchronous completion token pattern that is part of the **AsyncWaitSet** (p. 840) behavior.

A **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853) can be in one of the following states:

- `READY`: The completion token can be used to associate a new request. Calling **rti::core::cond::AsyncWaitSetCompletionToken::wait** (p. 855) on a ready completion token will return immediately with success. A ready completion token can only transition to the queued state.
- `QUEUED`: The completion token has an associated request that is pending processing. Calling **rti::core::cond::AsyncWaitSetCompletionToken::wait** (p. 855) on a queued completion token will block until the request completes or times out. A queued completion token can only transition to the processed state.
- `PROCESS`: The completion token has an associated request that has been processed but the application did not call **rti::core::cond::AsyncWaitSetCompletionToken::wait** (p. 855) yet. Calling **rti::core::cond::AsyncWaitSetCompletionToken::wait** (p. 855) on a processed completion token will return immediately with the return code result of processing the associated request. A processed completion token can transition to both ready or queued states.

8.22.2 AsyncWaitSetCompletionToken management

The same **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853) instance can be reused reused multiple times to associate a request and wait for its completion. Reusing is allowed only if the completion token is either in `READY` or `PROCESSED` state. Otherwise the **rti::core::cond::AsyncWaitSet** (p. 840) operation that associates the completion token will fail with **dds::core::PreconditionNotMetError** (p. 1941).

The completion token functionality can be viewed as a **rti::core::cond::AsyncWaitSet** (p. 840) internal detail from which your application should not need to know. In general, it is recommended to use the default flavor of **rti::core::cond::↩ AsyncWaitSet** (p. 840) operations that handle the internals of the completion tokens for you.

Nevertheless, if a completion token represents an expensive resource in your environment, your application may want to have full control of how and when completion tokens are created. It's for these reasons why is exposed as a public collaborator of the **rti::core::cond::AsyncWaitSet** (p. 840).

MT Safety:

Safe.

See also

rti::core::cond::AsyncWaitSet (p. 840)

8.22.3 Constructor & Destructor Documentation

8.22.3.1 AsyncWaitSetCompletionToken()

```
rti::core::cond::AsyncWaitSetCompletionToken::AsyncWaitSetCompletionToken (
              AsyncWaitSet & aws )  [inline]
```

Creates a new **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853).

All the created **rti::core::cond::AsyncWaitSetCompletionToken** (p. 853) must be deleted by calling rti::core::cond::↩ AsyncWaitSet::delete_completion_token.

See also

rti::core::cond::AsyncWaitSet::delete_completion_token

rti::core::cond::AsyncWaitSet::delete

8.22.4 Member Function Documentation

8.22.4.1 wait()

```
void rti::core::cond::AsyncWaitSetCompletionToken::wait (
            const  dds::core::Duration & max_wait =  dds::core::Duration::infinite() )  [inline]
```

Waits for the request associated to an operation made on the **rti::core::cond::AsyncWaitSet** (p. 840) to complete.

This operation will block the calling thread for a maximum amount of time specified by `max_wait` until the **rti::core::cond::AsyncWaitSet** (p. 840) request associated with this completion token completes.

If there is no timeout, upon return it is guaranteed that the request associated with this token completed. This operation may fail due to an error during the wait or while processing the associated request.

If this operation is called from within the context of one of the threads that comprise the thread pool of the **rti::core::cond::AsyncWaitSet** (p. 840), it will fail with **dds::core::PreconditionNotMetError** (p. 1941).

If the operation failed with **dds::core::TimeoutError** (p. 2473) your application can wait again on this completion token.

Parameters

max_wait	<<***in***>> *(p. 355)* Cannot be NULL. Maximum time to wait for the request associated to this completion token to complete before timeout.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

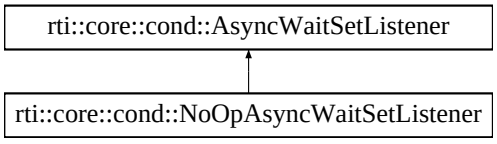
rti::core::cond::AsyncWaitSet (p. 840)

8.23 rti::core::cond::AsyncWaitSetListener Class Reference

Listener (p. 1637) for receiving event notifications related to the thread pool of the **AsyncWaitSet** (p. 840).

`#include <AsyncWaitSetListener.hpp>`

Inheritance diagram for rti::core::cond::AsyncWaitSetListener:



Public Member Functions

- virtual void **on_thread_spawned** (ThreadId thread_id)=0

  *Handles the spawning of each thread comprising the thread pool of the **rti::core::cond::AsyncWaitSet** (p. 840).*
- virtual void **on_thread_deleted** (ThreadId thread_id)=0

  *Handles the deletion of each thread comprising the thread pool of the **rti::core::cond::AsyncWaitSet** (p. 840).*
- virtual void **on_wait_timeout** (ThreadId thread_id)=0

  *Handles the wait timeout generated by the leader thread of the **rti::core::cond::AsyncWaitSet** (p. 840).*

8.23.1 Detailed Description

Listener (p. 1637) for receiving event notifications related to the thread pool of the **AsyncWaitSet** (p. 840).

8.23.2 Member Function Documentation

8.23.2.1 on_thread_spawned()

```
virtual void rti::core::cond::AsyncWaitSetListener::on_thread_spawned (
            ThreadId thread_id )  [pure virtual]
```

Handles the spawning of each thread comprising the thread pool of the **rti::core::cond::AsyncWaitSet** (p. 840).

Implemented in **rti::core::cond::NoOpAsyncWaitSetListener** (p. 1837).

8.23.2.2 on_thread_deleted()

```
virtual void rti::core::cond::AsyncWaitSetListener::on_thread_deleted (
            ThreadId thread_id )  [pure virtual]
```

Handles the deletion of each thread comprising the thread pool of the **rti::core::cond::AsyncWaitSet** (p. 840).

Implemented in **rti::core::cond::NoOpAsyncWaitSetListener** (p. 1837).

8.23.2.3 on_wait_timeout()

```
virtual void rti::core::cond::AsyncWaitSetListener::on_wait_timeout (
            ThreadId thread_id )  [pure virtual]
```

Handles the wait timeout generated by the leader thread of the **rti::core::cond::AsyncWaitSet** (p. 840).

Implemented in **rti::core::cond::NoOpAsyncWaitSetListener** (p. 1837).

8.24 rti::core::cond::AsyncWaitSetProperty Class Reference

Specifies the **rti::core::cond::AsyncWaitSet** (p. 840) behavior.

`#include <rti/core/cond/AsyncWaitSet.hpp>`

Public Member Functions

- **AsyncWaitSetProperty** ()

  Creates the default property.
- **AsyncWaitSetProperty** (**rti::core::cond::WaitSetProperty** &the_waitset_property, **dds::core::Duration** &the_wait_timeout, int32_t the_level, const std::string &the_thread_name_prefix, **rti::core::ThreadSettings** &the_thread_settings, int32_t the_thread_pool_size)

  Creates an instance with all the specified parameters.
- **rti::core::cond::WaitSetProperty waitset_property** () const

  Getter (see setter with the same name)
- **AsyncWaitSetProperty** & **waitset_property** (**rti::core::cond::WaitSetProperty** &the_waitset_property)

  *Specifies the behavior of the asynchronous wait behavior, which is equivalent to the wait mechanism of a **dds::core::cond::WaitSet** (p. 2617).*
- **dds::core::Duration wait_timeout** () const

  Getter (see setter with the same name)
- **AsyncWaitSetProperty** & **wait_timeout** (**dds::core::Duration** &the_wait_timeout)

  Asynchronous wait timeout.
- int32_t **level** () const

  Getter (see setter with the same name)
- **AsyncWaitSetProperty** & **level** (int32_t the_level)

  *Specifies the level of an **rti::core::cond::AsyncWaitSet** (p. 840).*
- const std::string **thread_name_prefix** () const

  Getter (see setter with the same name)
- **AsyncWaitSetProperty** & **thread_name_prefix** (const std::string &thr_thread_name_prefix)

  *Prefix used to compose the name of each thread that comprises the thread pool of the **rti::core::cond::AsyncWaitSet** (p. 840).*
- **rti::core::ThreadSettings thread_settings** () const

  Getter (see setter with the same name)
- **AsyncWaitSetProperty** & **thread_settings** (**rti::core::ThreadSettings** &the_thread_settings)

  ***rti::core::ThreadSettings** (p. 2449) for each thread comprising the thread pool of the **rti::core::cond::AsyncWaitSet** (p. 840).*
- int32_t **thread_pool_size** () const

  Getter (see setter with the same name)
- **AsyncWaitSetProperty** & **thread_pool_size** (int32_t the_level)

  *Number of threads that comprise the thread pool of the **rti::core::cond::AsyncWaitSet** (p. 840).*

8.24.1 Detailed Description

Specifies the **rti::core::cond::AsyncWaitSet** (p. 840) behavior.

This property allows configuring the behavior of the asynchronous wait and the **dds::core::cond::Condition** (p. 948) dispatch, as well as the parameters of the thread pool.

See also

rti::core::cond::WaitSetProperty (p. 2627)

rti::core::ThreadSettings (p. 2449)

8.24.2 Constructor & Destructor Documentation

8.24.2.1 AsyncWaitSetProperty() [1/2]

```
rti::core::cond::AsyncWaitSetProperty::AsyncWaitSetProperty ( )  [inline]
```

Creates the default property.

The default value for each field is documented in its corresponding setter.

8.24.2.2 AsyncWaitSetProperty() [2/2]

```
rti::core::cond::AsyncWaitSetProperty::AsyncWaitSetProperty (
             rti::core::cond::WaitSetProperty & the_waitset_property,
             dds::core::Duration & the_wait_timeout,
            int32_t the_level,
            const std::string & the_thread_name_prefix,
             rti::core::ThreadSettings & the_thread_settings,
            int32_t the_thread_pool_size )
```

Creates an instance with all the specified parameters.

8.24.3 Member Function Documentation

8.24.3.1 waitset_property() [1/2]

```
 rti::core::cond::WaitSetProperty rti::core::cond::AsyncWaitSetProperty::waitset_property ( )
const
```

Getter (see setter with the same name)

8.24.3.2 waitset_property() [2/2]

```
AsyncWaitSetProperty & rti::core::cond::AsyncWaitSetProperty::waitset_property (
            rti::core::cond::WaitSetProperty & the_waitset_property )
```

Specifies the behavior of the asynchronous wait behavior, which is equivalent to the wait mechanism of a **dds::core::cond::WaitSet** (p. 2617).

See also

rti::core::cond::WaitSetProperty (p. 2627)

8.24.3.3 wait_timeout() [1/2]

```
dds::core::Duration rti::core::cond::AsyncWaitSetProperty::wait_timeout ( ) const
```

Getter (see setter with the same name)

8.24.3.4 wait_timeout() [2/2]

```
AsyncWaitSetProperty & rti::core::cond::AsyncWaitSetProperty::wait_timeout (
            dds::core::Duration & the_wait_timeout )
```

Asynchronous wait timeout.

Specifies the maximum amount of time the leader thread of the **rti::core::cond::AsyncWaitSet** (p. 840) waits for an attached **dds::core::cond::Condition** (p. 948) to trigger before it wakes up.

Duration must be a value greater than zero.

See also

dds::core::cond::WaitSet::wait (p. 2621)

[default] dds::core::Duration::infinite() (p. 1451)

8.24.3.5 level() [1/2]

```
int32_t rti::core::cond::AsyncWaitSetProperty::level ( ) const
```

Getter (see setter with the same name)

8.24.3.6 level() [2/2]

```
AsyncWaitSetProperty & rti::core::cond::AsyncWaitSetProperty::level (
            int32_t the_level )
```

Specifies the level of an **rti::core::cond::AsyncWaitSet** (p. 840).

The level prevents an application to deadlock when it uses multiple **rti::core::cond::AsyncWaitSet** (p. 840) instances that call operations on each other from the context of one of their thread pool's thread.

Inside the context of one of these threads, the application can synchronize only with other **rti::core::cond::Async↩ WaitSet** (p. 840) of bigger level.

[default] 1

8.24.3.7 thread_name_prefix() [1/2]

```
const std::string rti::core::cond::AsyncWaitSetProperty::thread_name_prefix ( ) const
```

Getter (see setter with the same name)

8.24.3.8 thread_name_prefix() [2/2]

```
AsyncWaitSetProperty & rti::core::cond::AsyncWaitSetProperty::thread_name_prefix (
            const std::string & thr_thread_name_prefix )
```

Prefix used to compose the name of each thread that comprises the thread pool of the **rti::core::cond::AsyncWaitSet** (p. 840).

The composed name has the form:

`thread_name_prefix##[index]AWs` where [index] is an integer that identifies the thread relative to the **rti↩ ::core::cond::AsyncWaitSet** (p. 840).

If the string is empty, the default prefix will be used.

[default] Empty string (use default prefix)

8.24.3.9 thread_settings() [1/2]

```
rti::core::ThreadSettings rti::core::cond::AsyncWaitSetProperty::thread_settings ( ) const
```

Getter (see setter with the same name)

8.24.3.10 thread_settings() [2/2]

```
AsyncWaitSetProperty & rti::core::cond::AsyncWaitSetProperty::thread_settings (
            rti::core::ThreadSettings & the_thread_settings )
```

rti::core::ThreadSettings (p. 2449) for each thread comprising the thread pool of the **rti::core::cond::AsyncWaitSet** (p. 840).

Each thread within the pool is created with the same settings.

[default] Default thread settings values.

See also

rti::core::ThreadSettings (p. 2449)

8.24.3.11 thread_pool_size() [1/2]

```
int32_t rti::core::cond::AsyncWaitSetProperty::thread_pool_size ( ) const
```

Getter (see setter with the same name)

8.24.3.12 thread_pool_size() [2/2]

```
AsyncWaitSetProperty & rti::core::cond::AsyncWaitSetProperty::thread_pool_size (
            int32_t the_level )
```

Number of threads that comprise the thread pool of the **rti::core::cond::AsyncWaitSet** (p. 840).

Size must be equal to or greater than one.

[default] 1

8.25 rti::util::atomic_shared_ptr< T > Class Template Reference

Utility class to provide atomic operations on std::shared_ptr.

```
#include <atomic_shared_ptr.hpp>
```

8.25.1 Detailed Description

template<**class T**>
class rti::util::atomic_shared_ptr< **T** >

Utility class to provide atomic operations on std::shared_ptr.

This class provides a simple wrapper around std::shared_ptr to allow atomic operations such as store and load. It is useful in multi-threaded environments where shared pointers need to be accessed and modified safely.

8.26 rti::config::activity_context::AttributeKindMask Class Reference

The attributes used in the string representation of the Activity Context can be configured through this mask.

`#include <ActivityContext.hpp>`

Inherits std::bitset< 32 >.

Public Types

- typedef std::bitset< 32 > **MaskType**
  A typedef of std::bitset<32> for convenience.

Public Member Functions

- **AttributeKindMask** ()
  Not provide any attribute of the resource of the Activity Context.
- **AttributeKindMask** (uint64_t mask)
  *Construct a **AttributeKindMask*** (p. *863) from an integer.*
- **AttributeKindMask** (const **MaskType** &mask)
  *Construct a **AttributeKindMask*** (p. *863) from a MaskType object.*

Static Public Member Functions

- static const **AttributeKindMask** **guid_prefix** ()
  Provide the entity GUID prefix to the resource of the Activity Context.
- static const **AttributeKindMask** **topic** ()
  Provide the Topic to the resource of the Activity Context. The topic attribute is specified by "Topic".
- static const **AttributeKindMask** **type** ()
  Provide the data type to the resource of the Activity Context. The type attribute is specified by "Type".
- static const **AttributeKindMask** **entity_kind** ()
  Provide the entity kind to the resource of the Activity Context. The kind attribute is specified by "Entity".
- static const **AttributeKindMask** **domain_id** ()
  Provide the domain ID to the resource of the Activity Context. The domain attribute is specified by "Domain".
- static const **AttributeKindMask** **entity_name** ()
  Provide the entity name to the resource of the Activity Context. The name attribute is specified by "Name".
- static const **AttributeKindMask** **default_mask** ()
  Provide the default attributes of the resource of the Activity Context.
- static const **AttributeKindMask** **none** ()
  Not provide any attribute of the resource of the Activity Context.
- static const **AttributeKindMask** **all** ()
  Provide all the possibles attributes of the resource of the Activity Context.

8.26.1 Detailed Description

The attributes used in the string representation of the Activity Context can be configured through this mask.

This mask indicates what attributes of the resource are used when RTI Connext logs a message or when the Heap Monitoring utility saves statistics for a memory allocation.

8.26.2 Member Typedef Documentation

8.26.2.1 MaskType

```
typedef std::bitset<32>  rti::config::activity_context::AttributeKindMask::MaskType
```

A typedef of std::bitset<32> for convenience.

8.26.3 Constructor & Destructor Documentation

8.26.3.1 AttributeKindMask() [1/3]

```
rti::config::activity_context::AttributeKindMask::AttributeKindMask ( )  [inline]
```

Not provide any attribute of the resource of the Activity Context.

8.26.3.2 AttributeKindMask() [2/3]

```
rti::config::activity_context::AttributeKindMask::AttributeKindMask (
            uint64_t mask )  [inline], [explicit]
```

Construct a **AttributeKindMask** (p. 863) from an integer.

Parameters

mask	Value whose bits are copied to the bitset positions

8.26.3.3 AttributeKindMask() [3/3]

```
rti::config::activity_context::AttributeKindMask::AttributeKindMask (
            const  MaskType & mask )  [inline]
```

Construct a **AttributeKindMask** (p. 863) from a MaskType object.

Parameters

mask	A std::bitset<32> to construct this **AttributeKindMask** (p. 863) from

8.26.4 Member Function Documentation

8.26.4.1 guid_prefix()

```
static const  AttributeKindMask rti::config::activity_context::AttributeKindMask::guid_prefix ( )
[inline], [static]
```

Provide the entity GUID prefix to the resource of the Activity Context.

For example:

- For the following string representation of the context:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`
- The GUID prefix is 0X101A76B,0X79E5D71,0X50EE914. If the bit ActivityContextAttributeKind::GUID_PREFIX is not set, the string representation will not show the GUID prefix:
  `[:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`

8.26.4.2 topic()

```
static const  AttributeKindMask rti::config::activity_context::AttributeKindMask::topic ( )  [inline],
[static]
```

Provide the Topic to the resource of the Activity Context. The topic attribute is specified by "Topic".

For example:

- For the following string representation of the context:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`
- The Topic is "test." If the bit ActivityContextAttributeKind::TOPIC is not set, the string representation will not show the Topic:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Type=Foo,Domain=1}|Write]`

8.26.4.3 type()

```
static const AttributeKindMask rti::config::activity_context::AttributeKindMask::type ( ) [inline],
[static]
```

Provide the data type to the resource of the Activity Context. The type attribute is specified by "Type".

For example:

- For the following string representation of the context:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`
- The data type is "Foo." If the bit ActivityContextAttributeKind::TYPE is not set, the string representation will not show the data type:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Domain=1}|Write]`

8.26.4.4 entity_kind()

```
static const AttributeKindMask rti::config::activity_context::AttributeKindMask::entity_kind ( )
[inline], [static]
```

Provide the entity kind to the resource of the Activity Context. The kind attribute is specified by "Entity".

For example:

- For the following string representation of the context:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`
- The entity kind is "Writer." If the bit ActivityContextAttributeKind::ENTITY_KIND is not set, the string representation will not show the entity kind:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Topic=test,Type=Foo,Domain=1}|Write]`

8.26.4.5 domain_id()

```
static const AttributeKindMask rti::config::activity_context::AttributeKindMask::domain_id ( )
[inline], [static]
```

Provide the domain ID to the resource of the Activity Context. The domain attribute is specified by "Domain".

For example:

- For the following string representation of the context:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`
- The domain ID is "1." If the bit ActivityContextAttributeKind::DOMAIN_ID is not set, the string representation will not show the domain ID:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo}|Write]`

8.26.4.6 entity_name()

```
static const AttributeKindMask rti::config::activity_context::AttributeKindMask::entity_name ( )
[inline], [static]
```

Provide the entity name to the resource of the Activity Context. The name attribute is specified by "Name".

For example:

- For the following string representation of the context:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`
- The entity name is "testDataWriterName." If the bit ActivityContextAttributeKind::ENTITY_NAME is not set, the string representation will not show the entity name:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`

8.26.4.7 default_mask()

```
static const AttributeKindMask rti::config::activity_context::AttributeKindMask::default_mask ( )
[inline], [static]
```

Provide the default attributes of the resource of the Activity Context.

8.26.4.8 none()

```
static const AttributeKindMask rti::config::activity_context::AttributeKindMask::none ( ) [inline],
[static]
```

Not provide any attribute of the resource of the Activity Context.

8.26.4.9 all()

```
static const AttributeKindMask rti::config::activity_context::AttributeKindMask::all ( ) [inline],
[static]
```

Provide all the possibles attributes of the resource of the Activity Context.

8.27 rti::core::policy::Availability Class Reference

<<***extension***>> *(p. 116)* Configures data availability in the context of Collaborative DataWriters and Required Subscriptions

`#include <rti/core/policy/CorePolicy.hpp>`

Public Member Functions

- **Availability** ()

  Creates the default policy.
- **Availability** (bool the_enable_required_subscriptions, const **dds::core::Duration** &the_data_waiting_time, const **dds::core::Duration** &the_endpoint_waiting_time, const **rti::core::EndpointGroupSeq** &the_required↩_enpoint_groups)

  Creates a policy with all the paramters.
- **Availability** & **enable_required_subscriptions** (bool the_enable_required_subscriptions)

  <<***extension***>> *(p. 116) Enables support for required subscriptions in a* ***dds::pub::DataWriter*** *(p. 1138)*
- bool **enable_required_subscriptions** () const

  Getter (see setter with the same name)
- **Availability** & **max_data_availability_waiting_time** (const **dds::core::Duration** &the_data_waiting_time)

  <<***extension***>> *(p. 116) Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.*
- **dds::core::Duration max_data_availability_waiting_time** () const

  Getter (see setter with the same name)
- **Availability** & **max_endpoint_availability_waiting_time** (const **dds::core::Duration** &the_endpoint_↩waiting_time)

  <<***extension***>> *(p. 116) Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).*
- **dds::core::Duration max_endpoint_availability_waiting_time** () const

  Getter (see setter with the same name)
- **Availability** & **required_matched_endpoint_groups** (const **rti::core::EndpointGroupSeq** &the_required_↩enpoint_groups)

  <<***extension***>> *(p. 116) A sequence of endpoint groups.*
- **rti::core::EndpointGroupSeq required_matched_endpoint_groups** () const

  Getter (see setter with the same name)

8.27.1 Detailed Description

<<***extension***>> *(p. 116)* Configures data availability in the context of Collaborative DataWriters and Required Subscriptions

Entity:

dds::sub::DataReader (p. 976), **dds::pub::DataWriter** (p. 1138)

Properties:

RxO (p. 514) = NO
Changeable (p. 514) = **YES** (p. 514) (only on a **dds::pub::DataWriter** (p. 1138) except for the member **rti::core**↩**::policy::Availability::enable_required_subscriptions** (p. 871))

8.27.2 Usage

This QoS policy is used in the context of two features:

- Collaborative DataWriters
- Required Subscriptions

Collaborative DataWriters

The Collaborative DataWriters feature allows having multiple DataWriters publishing samples from a common logical data source. The DataReaders will combine the samples coming from the DataWriters in order to reconstruct the correct order at the source.

This QoS policy allows you to configure the ordering and combination process in the DataReader and can be used to support two different use cases:

- **Ordered delivery of samples in high-availability scenarios** One example of this is RTI Persistence **Service** (p. 2350). When a late-joining DataReader configured with **dds::core::policy::Durability** (p. 1433) set to **dds::core::policy::DurabilityKind_def::PERSISTENT** (p. 1442) or **dds::core::policy::DurabilityKind_def::↩ TRANSIENT** (p. 1442) joins a DDS domain, it will start receiving historical samples from multiple DataWriters. For example, if the original DataWriter is still alive, the newly created DataReader will receive samples from the original DataWriter and one or more RTI Persistence **Service** (p. 2350) DataWriters (PRSTDataWriters). This policy can be used to configure the sample ordering process on the DataReader.

- **Ordered delivery of samples in load-balanced scenarios** Multiple instances of the same application can work together to process and deliver samples. When the samples arrive through different data-paths out of order, the DataReader will be able to reconstruct the order at the destination. An example of this is when multiple instances of RTI Persistence **Service** (p. 2350) are used to persist the data. Persisting data to a database on disk can be a bottleneck for throughput. You can improve scalability and performance by dividing the workload across different instances of RTI Persistence **Service** (p. 2350) that use different databases. For example, samples larger than 10 are persisted by Persistence **Service** (p. 2350) 1, samples less than or equal to 10 are persisted by Persistence **Service** (p. 2350) 2.

- **Ordered delivery of samples with Group Ordered Access** This policy can also be used to configure the sample ordering process when the Subscriber is configured with **dds::core::policy::Presentation** (p. 1942) access_↩ scope set to **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951). In this case, the Subscriber must deliver in order the samples published by a group of DataWriters that belong to the same Publisher and have access_scope set to **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951).

Each sample published in a DDS domain for a given logical data source is uniquely identified by a pair (virtual GUID, virtual sequence number). Samples from the same data source (same virtual GUID) can be published by different DataWriters. A DataReader will deliver a sample (VGUIDn, VSNm) to the application if one of the following conditions is satisfied:

- (VGUIDn, VSNm-1) has already been delivered to the application.
- All the known DataWriters publishing VGUIDn have announced that they do not have (VGUIDn, VSNm-1).

- None of the known DataWriters publishing GUIDn have announced potential availability of (VGUIDn, VSNm-1) and both timeouts in this QoS policy have expired.

A DataWriter announces potential availability of samples by using virtual heartbeats (HBs).

When **dds::core::policy::Presentation::access_scope** (p. 1947) is set to **dds::core::policy::PresentationAccessScopeKind_def::TOPIC** (p. 1951) or **dds::core::policy::PresentationAccessScopeKind_def::INSTANCE** (p. 1951), the virtual HB contains information about the samples contained in the **dds::pub::DataWriter** (p. 1138) history.

When **dds::core::policy::Presentation::access_scope** (p. 1947) is set to **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951), the virtual HB contains information about all DataWriters in the **dds::pub::Publisher** (p. 1993).

The frequency at which virtual HBs are sent is controlled by the protocol parameters **rti::core::policy::RtpsReliableWriterProtocol::virtual_heartbeat_period** (p. 2237) and **rti::core::policy::RtpsReliableWriterProtocol::samples_per_virtual_heartbeat** (p. 2238).

Required Subscriptions

In the context of Required Subscriptions, this QoS policy can be used to configure a set of Required Subscriptions on a **dds::pub::DataWriter** (p. 1138).

Required subscriptions are preconfigured, named subscriptions that may leave and subsequently rejoin the network from time to time, at the same or different physical locations. Any time a required subscription is disconnected, any samples that would have been delivered to it are stored for delivery if and when the subscription rejoins the network.

8.27.3 Consistency

For a DataWriter, the setting of **AVAILABILITY** (p. 516) must be set consistenly with that of the **RELIABILITY** (p. 544) and **DURABILITY** (p. 529).

If **rti::core::policy::Availability::enable_required_subscriptions** (p. 871) is set to true, **dds::core::policy::Reliability::kind** (p. 2153) must be set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157), **dds::core::policy::Durability** (p. 1433) must be set to a value different than **dds::core::policy::DurabilityKind_def::VOLATILE** (p. 1442), and **dds::core::policy::Durability::writer_depth** (p. 1439) must be set to either **rti::core::policy::auto_writer_depth()** (p. 531) or **dds::core::LENGTH_UNLIMITED** (p. 438).

8.27.4 Constructor & Destructor Documentation

8.27.4.1 Availability() [1/2]

```
rti::core::policy::Availability::Availability ( )  [inline]
```

Creates the default policy.

8.27.4.2 Availability() [2/2]

```
rti::core::policy::Availability::Availability (
            bool the_enable_required_subscriptions,
            const  dds::core::Duration & the_data_waiting_time,
            const  dds::core::Duration & the_endpoint_waiting_time,
            const  rti::core::EndpointGroupSeq & the_required_enpoint_groups )
```

Creates a policy with all the paramters.

8.27.5 Member Function Documentation

8.27.5.1 enable_required_subscriptions() [1/2]

```
Availability & rti::core::policy::Availability::enable_required_subscriptions (
            bool the_enable_required_subscriptions )
```

<<***extension***>> *(*p. *116)* Enables support for required subscriptions in a **dds::pub::DataWriter** (p. 1138)

[default] false

Categories

Immutable (p. 243), **Reliability** (p. 222), **Durability** (p. 184), **Data Availability** (p. 167)

8.27.5.2 enable_required_subscriptions() [2/2]

```
bool rti::core::policy::Availability::enable_required_subscriptions ( ) const
```

Getter (see setter with the same name)

8.27.5.3 max_data_availability_waiting_time() [1/2]

```
Availability & rti::core::policy::Availability::max_data_availability_waiting_time (
            const  dds::core::Duration & the_data_waiting_time )
```

<<***extension***>> *(p. 116)* Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.

Collaborative DataWriters

A sample identified by (VGUIDn, VSNm) will be delivered to the application if this timeout expires for the sample and the following two conditions are satisfied:

- None of the known DataWriters publishing VGUIDn have announced potential availability of (VGUIDn, VSNm-1).
- The DataWriters for all the endpoint groups specified in **required_matched_endpoint_groups** (p. 873) have been discovered or **max_endpoint_availability_waiting_time** (p. 872) has expired.

Required Subscriptions

This field is not applicable to Required Subscriptions.

[default] **dds::core::Duration::automatic()** (p. 1451) (**dds::core::Duration::infinite()** (p. 1451) for **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951). Otherwise, 0 seconds)

[range] [0, **dds::core::Duration::infinite()** (p. 1451)], **dds::core::Duration::automatic()** (p. 1451)

Categories

Mutable (p. 280), **Reliability** (p. 222), **Durability** (p. 184), **Data Availability** (p. 167)

8.27.5.4 max_data_availability_waiting_time() [2/2]

```
dds::core::Duration rti::core::policy::Availability::max_data_availability_waiting_time ( ) const
```

Getter (see setter with the same name)

8.27.5.5 max_endpoint_availability_waiting_time() [1/2]

```
Availability & rti::core::policy::Availability::max_endpoint_availability_waiting_time (
            const  dds::core::Duration & the_endpoint_waiting_time )
```

<<***extension***>> *(p. 116)* Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).

Collaborative DataWriters

The set of endpoint groups that are required to provide samples for a data source can be configured using **required_matched_endpoint_groups** (p. 873).

A non-consecutive sample identified by (VGUIDn, VSNm) cannot be delivered to the application unless DataWriters for all the endpoint groups in **required_matched_endpoint_groups** (p. 873) are discovered or this timeout expires.

Required Subscriptions

This field is not applicable to Required Subscriptions.

[default] **dds::core::Duration::automatic()** (p. 1451) (**dds::core::Duration::infinite()** (p. 1451) for **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951). Otherwise, 0 seconds)

[range] [0, **dds::core::Duration::infinite()** (p. 1451)], **dds::core::Duration::automatic()** (p. 1451)

Categories

Mutable (p. 280), **Reliability** (p. 222), **Durability** (p. 184), **Data Availability** (p. 167)

8.27.5.6 max_endpoint_availability_waiting_time() [2/2]

```
dds::core::Duration rti::core::policy::Availability::max_endpoint_availability_waiting_time ( )
const
```

Getter (see setter with the same name)

8.27.5.7 required_matched_endpoint_groups() [1/2]

```
Availability & rti::core::policy::Availability::required_matched_endpoint_groups (
            const  rti::core::EndpointGroupSeq & the_required_enpoint_groups )
```

<<***extension***>> *(p. 116)* A sequence of endpoint groups.

Collaborative DataWriters

In the context of Collaborative DataWriters, it specifies the set of endpoint groups that are expected to provide samples for the same data source.

The quorum count in a group represents the number of DataWriters that must be discovered for that group before the DataReader is allowed to provide non consecutive samples to the application.

A DataWriter becomes a member of an endpoint group by configuring the role_name in **rti::core::policy::EntityName** (p. 1530).

Required Subscriptions

In the context of Required Subscriptions, it specifies the set of Required Subscriptions on a **dds::pub::DataWriter** (p. 1138).

Each Required Subscription is specified by a name and a quorum count.

The quorum count represents the number of DataReaders that have to acknowledge the sample before it can be considered fully acknowledged for that Required Subscription.

A DataReader is associated with a Required Subscription by configuring the role_name in **rti::core::policy::Entity↩ Name** (p. 1530).

[default] Empty sequence

Categories

Mutable (p. 280), **Reliability** (p. 222), **Durability** (p. 184), **Data Availability** (p. 167)

8.27.5.8 required_matched_endpoint_groups() [2/2]

```
 rti::core::EndpointGroupSeq rti::core::policy::Availability::required_matched_endpoint_groups ( )
const
```

Getter (see setter with the same name)

8.28 dds::core::basic_string< CharType, Allocator > Class Template Reference

<<***value-type***>> *(p. 350)* A string convertible to std::string and with similar functionality

`#include <dds/core/String.hpp>`

Public Member Functions

- **basic_string** ()
  Creates an empty string.
- **basic_string** (const **basic_string** &other)
  Copy constructor.
- **basic_string** (const CharType *other_str)
  Constructor from C string.
- **basic_string** (const std::basic_string< CharType > &std_string)
  Constructor from `std::basic_string`.
- **basic_string** (size_t the_size)
  Creates a string of a given size, initialized to '\0'.
- **basic_string** (**basic_string** &&other) OMG_NOEXCEPT
  <<***C++11***>> *(p. 353) Move constructor*
- const CharType * **c_str** () const
  Gets the underlying C string.
- size_type **size** () const
  Gets the size.
- **basic_string** & **operator=** (const **basic_string** &other)
  Assignment operator.
- **basic_string** & **operator=** (**basic_string** &&other) OMG_NOEXCEPT
  <<***C++11***>> *(p. 353) Move-assignment operator*
- bool **operator==** (const **basic_string** &other) const
  Returns if two strings are equal.
- bool **operator!=** (const **basic_string** &other) const
  Returns if two strings are different.
- std::basic_string< CharType > **to_std_string** () const
  *Creates a `std::basic_string` from this **`dds::core::basic_string`** (p. `875`).*
- **operator std::basic_string< CharType >** () const
  *Creates a `std::string` from this **`dds::core::string`** (p. `435`).*

Related Functions

(Note that these are not member functions.)

- template<typename CharType , typename Allocator >
  std::ostream & **operator<<** (std::ostream &out, const **basic_string**< CharType, Allocator > &the_string)
  Prints the string.

8.28.1 Detailed Description

template<typename CharType, typename Allocator>
class dds::core::basic_string< CharType, Allocator >

<<***value-type***>> *(p. 350)* A string convertible to std::string and with similar functionality

In many cases, for performance reasons and other implementation requirements, the RTI Connext API uses **dds::core::string** (p. 435) instead of std::string. The most significant case is the C++ types that rtiddsgen generates from **IDL** (p. 606).

A **dds::core::string** (p. 435) provides a subset of the functionality of a `std::string`. It also provides automatic conversion to and from `std::string`.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 basic_string() [1/6]

```
template<typename CharType , typename Allocator >
dds::core::basic_string< CharType, Allocator >::basic_string ( )  [inline]
```

Creates an empty string.

8.28.2.2 basic_string() [2/6]

```
template<typename CharType , typename Allocator >
dds::core::basic_string< CharType, Allocator >::basic_string (
            const  basic_string< CharType, Allocator > & other )  [inline]
```

Copy constructor.

References **dds::core::basic_string< CharType, Allocator >::c_str()**, and **dds::core::basic_string< CharType, Allocator >::size()**.

8.28.2.3 basic_string() [3/6]

```
template<typename CharType , typename Allocator >
dds::core::basic_string< CharType, Allocator >::basic_string (
            const CharType * other_str )  [inline]
```

Constructor from C string.

8.28.2.4 basic_string() [4/6]

```
template<typename CharType , typename Allocator >
dds::core::basic_string< CharType, Allocator >::basic_string (
            const std::basic_string< CharType > & std_string )  [inline]
```

Constructor from `std::basic_string`.

8.28.2.5 basic_string() [5/6]

```
template<typename CharType , typename Allocator >
dds::core::basic_string< CharType, Allocator >::basic_string (
            size_t the_size )  [inline], [explicit]
```

Creates a string of a given size, initialized to '\0'.

8.28.2.6 basic_string() [6/6]

```
template<typename CharType , typename Allocator >
dds::core::basic_string< CharType, Allocator >::basic_string (
             basic_string< CharType, Allocator > && other )  [inline]
```

<<***C++11***>> *(p. 353)* Move constructor

References **dds::core::Duration::swap()**.

8.28.3 Member Function Documentation

8.28.3.1 c_str()

```
template<typename CharType , typename Allocator >
const CharType *  dds::core::basic_string< CharType, Allocator >::c_str ( ) const  [inline]
```

Gets the underlying C string.

Referenced by **dds::core::basic_string< CharType, Allocator >::basic_string()**, and **dds::core::basic_string< CharType, Allocator >::operator<<()**.

8.28.3.2 size()

```
template<typename CharType , typename Allocator >
size_type  dds::core::basic_string< CharType, Allocator >::size ( ) const  [inline]
```

Gets the size.

Referenced by **dds::core::basic_string< CharType, Allocator >::basic_string()**, **dds::core::basic_string< CharType, Allocator >::operator=()**, and **dds::core::basic_string< CharType, Allocator >::operator==()**.

8.28.3.3 operator=() [1/2]

```
template<typename CharType , typename Allocator >
basic_string &  dds::core::basic_string< CharType, Allocator >::operator= (
             const  basic_string< CharType, Allocator > & other )  [inline]
```

Assignment operator.

References **dds::core::basic_string< CharType, Allocator >::size()**, and **dds::core::Duration::swap()**.

8.28.3.4 operator=() [2/2]

```
template<typename CharType , typename Allocator >
basic_string &  dds::core::basic_string< CharType, Allocator >::operator= (
              basic_string< CharType, Allocator > && other )  [inline]
```

<<***C++11***>> *(p. 353)* Move-assignment operator

References **dds::core::Duration::swap()**.

8.28.3.5 operator==()

```
template<typename CharType , typename Allocator >
bool  dds::core::basic_string< CharType, Allocator >::operator== (
             const  basic_string< CharType, Allocator > & other ) const  [inline]
```

Returns if two strings are equal.

References **dds::core::basic_string< CharType, Allocator >::size()**.

8.28.3.6 operator"!=()

```
template<typename CharType , typename Allocator >
bool  dds::core::basic_string< CharType, Allocator >::operator!= (
            const  basic_string< CharType, Allocator > & other ) const  [inline]
```

Returns if two strings are different.

8.28.3.7 to_std_string()

```
template<typename CharType , typename Allocator >
std::basic_string< CharType >  dds::core::basic_string< CharType, Allocator >::to_std_string ( )
const  [inline]
```

Creates a `std::basic_string` from this **dds::core::basic_string** (p. 875).

8.28.3.8 operator std::basic_string< CharType >()

```
template<typename CharType , typename Allocator >
dds::core::basic_string< CharType, Allocator >::operator std::basic_string< CharType > ( ) const
[inline]
```

Creates a `std::string` from this **dds::core::string** (p. 435).

8.28.4 Friends And Related Function Documentation

8.28.4.1 operator<<()

```
template<typename CharType , typename Allocator >
std::ostream & operator<< (
            std::ostream & out,
            const  basic_string< CharType, Allocator > & the_string )  [related]
```

Prints the string.

References **dds::core::basic_string< CharType, Allocator >::c_str()**.

8.29 rti::core::policy::Batch Class Reference

<<***extension***>> *(p. 116)* Allows a **dds::pub::DataWriter** (p. 1138) to batch multiple samples into a single network packet to increase throughput.

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **Batch** ()

  Creates the default policy (batching is disabled)
- **Batch** & **enable** (bool the_enable)

  Enables or disables batching.
- bool **enable** () const

  Returns whether batching is enabled or not.
- **Batch** & **max_data_bytes** (int32_t the_max_data_bytes)

  Sets the maximum cumulative length of all serialized samples in a batch.
- int32_t **max_data_bytes** () const

  Getter (see setter with the same name)
- **Batch** & **max_samples** (int32_t the_max_samples)

  Sets the maximum number of samples in a batch.
- int32_t **max_samples** () const

  Getter (see setter with the same name)
- **Batch** & **max_flush_delay** (const **dds::core::Duration** &the_max_flush_delay)

  Sets the maximum delay after which a batch is flushed.
- **dds::core::Duration** **max_flush_delay** () const

  Getter (see setter with the same name)
- **Batch** & **source_timestamp_resolution** (const **dds::core::Duration** &the_source_timestamp_resolution)

  Sets the timestamp resolution of the samples in a batch.
- **dds::core::Duration** **source_timestamp_resolution** () const

  Getter (see setter with the same name)
- **Batch** & **thread_safe_write** (bool the_thread_safe_write)

  Indicates if the write operation needs to be thread-safe.
- bool **thread_safe_write** () const

  Getter (see setter with the same name)

Static Public Member Functions

- static **Batch** **Enabled** ()

  *Returns an instance that enables batching with the default **max_samples()** (p. 883) and **max_data_bytes()** (p. 882).*
- static **Batch** **Disabled** ()

  Returns an instance that disables batching.
- static **Batch** **EnabledWithMaxDataBytes** (int32_t the_max_bytes)

  *Returns an instance that enables batching with **max_data_bytes()** (p. 882)*
- static **Batch** **EnabledWithMaxSamples** (int32_t the_max_samples)

  *Returns an instance that enables batching with **max_samples()** (p. 883)*

8.29.1 Detailed Description

<<***extension***>> *(p. 116)* Allows a **dds::pub::DataWriter** (p. 1138) to batch multiple samples into a single network packet to increase throughput.

This QoS policy configures the ability of the middleware to collect multiple user data samples to be sent in a single network packet, to take advantage of the efficiency of sending larger packets and thus increase effective throughput.

This QoS policy can be used to dramatically increase effective throughput for small data samples. Usually, throughput for small samples (size $<$ 2048 bytes) is limited by CPU capacity and not by network bandwidth. Batching many smaller samples to be sent in a single large packet will increase network utilization, and thus throughput, in terms of samples per second.

Entity:

dds::pub::DataWriter (p. 1138)

Properties:

RxO (p. 514) = NO
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

8.29.2 Constructor & Destructor Documentation

8.29.2.1 Batch()

```
rti::core::policy::Batch::Batch ( )  [inline]
```

Creates the default policy (batching is disabled)

8.29.3 Member Function Documentation

8.29.3.1 Enabled()

```
static  Batch rti::core::policy::Batch::Enabled ( )  [inline], [static]
```

Returns an instance that enables batching with the default **max_samples()** (p. 883) and **max_data_bytes()** (p. 882).

8.29.3.2 Disabled()

```
static Batch rti::core::policy::Batch::Disabled ( ) [inline], [static]
```

Returns an instance that disables batching.

8.29.3.3 EnabledWithMaxDataBytes()

```
static Batch rti::core::policy::Batch::EnabledWithMaxDataBytes (
            int32_t the_max_bytes ) [inline], [static]
```

Returns an instance that enables batching with **max_data_bytes()** (p. 882)

For example:

```
using namespace rti::core::policy;
writer_qos « Batch::EnabledWithMaxDataBytes(2048);
```

8.29.3.4 EnabledWithMaxSamples()

```
static Batch rti::core::policy::Batch::EnabledWithMaxSamples (
            int32_t the_max_samples ) [inline], [static]
```

Returns an instance that enables batching with **max_samples()** (p. 883)

8.29.3.5 enable() [1/2]

```
Batch & rti::core::policy::Batch::enable (
            bool the_enable )
```

Enables or disables batching.

Referenced by **max_data_bytes()**, and **max_flush_delay()**.

8.29.3.6 enable() [2/2]

```
bool rti::core::policy::Batch::enable ( ) const
```

Returns whether batching is enabled or not.

8.29.3.7 max_data_bytes() [1/2]

```
Batch & rti::core::policy::Batch::max_data_bytes (
            int32_t the_max_data_bytes )
```

Sets the maximum cumulative length of all serialized samples in a batch.

A batch is flushed automatically when this maximum is reached.

max_data_bytes does not include the meta data associated with the batch samples. Each sample has at least 8 bytes of meta data containing information such as the timestamp and sequence number. The meta data can be as large as 52 bytes for keyed topics and 20 bytes for unkeyed topics.

Note: Batches must contain whole samples. If a new batch is started and its initial sample causes the serialized size to exceed max_data_bytes, RTI Connext will send the sample in a single batch.

[default] 1024

[range] [1,**dds::core::LENGTH_UNLIMITED** (p. 438)]

8.29.4 Consistency

The setting of **rti::core::policy::Batch::max_data_bytes** (p. 882) must be consistent with **rti::core::policy::Batch↩ ::max_samples** (p. 883). For these two values to be consistent, they cannot be both **dds::core::LENGTH_UNLIMITED** (p. 438).

Categories

Immutable (p. 243), **Batching** (p. 161), **Throughput** (p. 236)

Referenced by **max_flush_delay()**.

8.29.4.1 max_data_bytes() [2/2]

```
int32_t rti::core::policy::Batch::max_data_bytes ( ) const
```

Getter (see setter with the same name)

References **enable()**.

8.29.4.2 max_samples() [1/2]

```
Batch & rti::core::policy::Batch::max_samples (
            int32_t the_max_samples )
```

Sets the maximum number of samples in a batch.

A batch is flushed automatically when this maximum is reached.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1,**dds::core::LENGTH_UNLIMITED** (p. 438)]

8.29.5 Consistency

The setting of **rti::core::policy::Batch::max_samples** (p. 883) must be consistent with **rti::core::policy::Batch↩ ::max_data_bytes** (p. 882). For these two values to be consistent, they cannot be both **dds::core::LENGTH_↩ UNLIMITED** (p. 438).

Categories

Immutable (p. 243), **Batching** (p. 161), **Throughput** (p. 236)

8.29.5.1 max_samples() [2/2]

```
int32_t rti::core::policy::Batch::max_samples ( ) const
```

Getter (see setter with the same name)

8.29.5.2 max_flush_delay() [1/2]

```
Batch & rti::core::policy::Batch::max_flush_delay (
            const  dds::core::Duration & the_max_flush_delay )
```

Sets the maximum delay after which a batch is flushed.

A batch is flushed automatically after the delay specified by this parameter.

The delay is measured from the time the first sample in the batch is written by the application.

[default] dds::core::Duration::infinite() (p. 1451)

[range] [0,**dds::core::Duration::infinite()** (p. 1451)]

8.29.6 Consistency

The setting of **rti::core::policy::Batch::max_flush_delay** (p. 884) must be consistent with **rti::core::policy::AsynchronousPublisher::disable_asynchronous_batch** (p. 836) and **rti::core::policy::Batch::thread_safe_write** (p. 886). If the delay is different than **dds::core::Duration::infinite()** (p. 1451), **rti::core::policy::AsynchronousPublisher::disable_asynchronous_batch** (p. 836) must be set to false and **rti::core::policy::Batch::thread_safe_write** (p. 886) must be set to true.

Categories

Immutable (p. 243), **Batching** (p. 161), **Throughput** (p. 236)

References **enable()**, and **max_data_bytes()**.

8.29.6.1 max_flush_delay() [2/2]

```
dds::core::Duration rti::core::policy::Batch::max_flush_delay ( ) const
```

Getter (see setter with the same name)

8.29.6.2 source_timestamp_resolution() [1/2]

```
Batch & rti::core::policy::Batch::source_timestamp_resolution (
            const  dds::core::Duration & the_source_timestamp_resolution )
```

Sets the timestamp resolution of the samples in a batch.

The value of this field determines how the source timestamp is associated with the samples in a batch.

A sample written with timestamp 't' inherits the source timestamp 't2' associated with the previous sample unless ('t' - 't2') $>$ source_timestamp_resolution.

If source_timestamp_resolution is set to **dds::core::Duration::infinite()** (p. 1451), every sample in the batch will share the source timestamp associated with the first sample.

If source_timestamp_resolution is set to zero, every sample in the batch will contain its own source timestamp corresponding to the moment when the sample was written.

The performance of the batching process is better when source_timestamp_resolution is set to **dds::core::Duration::infinite()** (p. 1451).

[default] dds::core::Duration::infinite() (p. 1451)

[range] [0,**dds::core::Duration::infinite()** (p. 1451)]

8.29.7 Consistency

The setting of **rti::core::policy::Batch::source_timestamp_resolution** (p. 885) must be consistent with **rti::core::policy::Batch::thread_safe_write** (p. 886). If **rti::core::policy::Batch::thread_safe_write** (p. 886) is set to false, **rti::core::policy::Batch::source_timestamp_resolution** (p. 885) must be set to **dds::core::Duration::infinite()** (p. 1451).

Categories

Immutable (p. 243), **Batching** (p. 161), **Throughput** (p. 236), **Data Availability** (p. 167)

8.29.7.1 source_timestamp_resolution() [2/2]

```
dds::core::Duration rti::core::policy::Batch::source_timestamp_resolution ( ) const
```

Getter (see setter with the same name)

8.29.7.2 thread_safe_write() [1/2]

```
Batch & rti::core::policy::Batch::thread_safe_write (
            bool the_thread_safe_write )
```

Indicates if the write operation needs to be thread-safe.

If this parameter is set to true, multiple threads can call write on the **dds::pub::DataWriter** (p. 1138) concurrently.

[default] true

8.29.8 Consistency

The setting of **rti::core::policy::Batch::thread_safe_write** (p. 886) must be consistent with **rti::core::policy::Batch::source_timestamp_resolution** (p. 885). If **rti::core::policy::Batch::thread_safe_write** (p. 886) is set to false, **rti::core::policy::Batch::source_timestamp_resolution** (p. 885) must be set to **dds::core::Duration::infinite()** (p. 1451).

Categories

Immutable (p. 243), **Batching** (p. 161), **Throughput** (p. 236), **Concurrency** (p. 164)

8.29.8.1 thread_safe_write() [2/2]

```
bool rti::core::policy::Batch::thread_safe_write ( ) const
```

Getter (see setter with the same name)

8.30 rti::core::bounded_sequence< T, MaxLength > Class Template Reference

<<***value-type***>> *(*p. *350)* A bounded sequence of elements

```
#include <rti/core/BoundedSequence.hpp>
```

Public Types

- typedef std::vector< T > **vector_type**
  The type of the underlying vector.
- typedef vector_type::value_type **value_type**
  The type of the elements, `T`.
- typedef vector_type::allocator_type **allocator_type**
  The allocator type of the underlying vector.
- typedef vector_type::size_type **size_type**
  The size type.
- typedef vector_type::difference_type **difference_type**
  The difference type.
- typedef vector_type::reference **reference**
  The reference type.
- typedef vector_type::const_reference **const_reference**
  The const reference type.
- typedef vector_type::pointer **pointer**
  The pointer type.
- typedef vector_type::const_pointer **const_pointer**
  The const pointer type.
- typedef vector_type::iterator **iterator**
  The iterator type.
- typedef vector_type::const_iterator **const_iterator**
  The const iterator type.
- typedef vector_type::reverse_iterator **reverse_iterator**
  The reverse iterator type.
- typedef vector_type::const_reverse_iterator **const_reverse_iterator**
  The const reverse iterator type.

Public Member Functions

- **bounded_sequence** ()

  *Creates an empty **bounded_sequence** (p. 887).*
- **bounded_sequence** (const **bounded_sequence** &other)

  Copy constructor.
- template<size_t M>
  bounded_sequence (const **bounded_sequence**< T, M > &other)

  Copies a bounded sequence with a different bound M.
- **bounded_sequence** (**size_type** count)

  *Create a **bounded_sequence** (p. 887) with a number of default-constructed elements.*
- **bounded_sequence** (**size_type** count, const **value_type** &value)

  *Create a **bounded_sequence** (p. 887) with a number of elements.*
- template<typename InputIt , typename Category = typename std::iterator_traits<InputIt>::iterator_category>
  bounded_sequence (InputIt &&first, InputIt &&last)

  *Create a **bounded_sequence** (p. 887) from a range of elements.*
- **bounded_sequence** (const std::vector< T > &v)

  *Create a **bounded_sequence** (p. 887) copying the elements of a std::vector.*
- **bounded_sequence** (**bounded_sequence** &&other) OMG_NOEXCEPT

  *<<**C++11**>> (p. 353) Move constructor*
- **bounded_sequence** (std::vector< T > &&v)

  *<<**C++11**>> (p. 353) Creates a **bounded_sequence** (p. 887) by moving a std::vector*
- **bounded_sequence** & **operator=** (const **bounded_sequence** &other)

  Assignment operator.
- template<typename U , size_t M>
  bounded_sequence & **operator=** (const **bounded_sequence**< U, M > &other)

  Copies the elements of a sequence with a different bound.
- **bounded_sequence** & **operator=** (const std::vector< T > &v)

  Copies the elements of a std::vector.
- **bounded_sequence** & **operator=** (**bounded_sequence** &&other) OMG_NOEXCEPT

  *<<**C++11**>> (p. 353) Move-assignment operator*
- **bounded_sequence** & **operator=** (std::vector< T > &&v)

  *<<**C++11**>> (p. 353) Moves a std::vector into this **bounded_sequence** (p. 887)*
- **bounded_sequence** & **operator=** (std::initializer_list< T > l)

  *<<**C++11**>> (p. 353) Copies the elements in an initializer_list*
- **reference operator[ ]** (**size_type** pos)

  Index access.
- **const_reference operator[ ]** (**size_type** pos) const

  Index access.
- **reference at** (**size_type** pos)

  Index access with bounds check.
- **const_reference at** (**size_type** pos) const

  Index access with bounds check.
- **reference front** ()

  Returns a reference to the first element.
- **const_reference front** () const

  Returns a const reference to the first element.
- **reference back** ()

Returns a reference to the last element.

- **const_reference back** () const

  Returns a const reference to the last element.

- **pointer data** ()

  Returns a pointer to the underlying data buffer.

- **const_pointer data** () const

  Returns a pointer to the underlying data buffer.

- **iterator begin** () OMG_NOEXCEPT

  Returns an iterator to the first element.

- **const_iterator begin** () const OMG_NOEXCEPT

  Returns an const iterator to the first element.

- **const_iterator cbegin** () const OMG_NOEXCEPT

  Returns an const iterator to the first element.

- **iterator end** () OMG_NOEXCEPT

  Returns an iterator to one past the last element.

- **const_iterator end** () const OMG_NOEXCEPT

  Returns an const iterator to one past the last element.

- **const_iterator cend** () const OMG_NOEXCEPT

  Returns a const iterator to one past the last element.

- **reverse_iterator rbegin** () OMG_NOEXCEPT

  Returns an iterator to the reverse-beginning of the sequence.

- **const_reverse_iterator rbegin** () const OMG_NOEXCEPT

  Returns a const iterator to the reverse-beginning of the sequence.

- **const_reverse_iterator crbegin** () const OMG_NOEXCEPT

  Returns a const iterator to the reverse-beginning of the sequence.

- **reverse_iterator rend** () OMG_NOEXCEPT

  Returns an iterator to the reverse-end of the sequence.

- **const_reverse_iterator crend** () const OMG_NOEXCEPT

  Returns a const iterator to the reverse-end of the sequence.

- bool **empty** () const OMG_NOEXCEPT

  Returns whether there are no elements.

- **size_type size** () const OMG_NOEXCEPT

  Returns the number of elements.

- **size_type max_size** () const OMG_NOEXCEPT

  Returns `MaxLength`*.*

- void **reserve** (**size_type** new_capacity)

  Pre-allocates elements up to new_capacity.

- **size_type capacity** () const OMG_NOEXCEPT

  Returns the current capacity.

- void **clear** () OMG_NOEXCEPT

  Removes the elements.

- void **shrink_to_fit** ()

  Destroys any extra elements reserved above the current size.

- **iterator insert** (**iterator** pos, const T &value)

  Inserts a new element in a position specified by an iterator.

- **iterator erase** (**iterator** pos)

  Erases an element specified by an iterator.

- void **push_back** (const T &value)

  Adds a new element at the end.
- void **push_back** (T &&value)

  Adds a new element at the end by moving it.
- void **pop_back** ()

  Removes the last element.
- void **resize** (**size_type** count)

  Resizes the container to contain count elements.
- void **resize** (**size_type** count, const T &value)

  Resizes the container to contain count elements.
- void **swap** (**bounded_sequence** &other) OMG_NOEXCEPT

  Swap the contents of two sequences.

8.30.1 Detailed Description

template<typename T, size_t MaxLength>
class rti::core::bounded_sequence< T, MaxLength >

<<***value-type***>> *(*p. *350)* A bounded sequence of elements

Template Parameters

T	The element type
MaxLength	The maximum number of elements

IDL bounded sequences map to this type, as described in **Working with IDL types** (p. 606).

This type has a interface similar to std::vector, but it differs in two aspects:

- It enforces its bound, `MaxLength`, in its operations. Operations that would make the sequence grow beyond the limit throw **`dds::core::PreconditionNotMetError`** (p. 1941).

- it implements a different element life-cycle strategy that optimizes the performance of the internal **dds::sub::DataReader** (p. 976) and **dds::pub::DataWriter** (p. 1138) serialization operations. Just like a std::vector, a **bounded_sequence** (p. 887) has a size and a capacity. But when capacity > size, the extra elements are default-constructed, unlike a std::vector where the extra elements are uninitialized memory. Functions that reduce the size of the sequence do not destroy the elements that they remove. Only **shrink_to_fit()** (p. 902) does.

See also

Working with IDL types (p. 606).

Examples

Foo.hpp.

8.30.2 Member Typedef Documentation

8.30.2.1 vector_type

```
template<typename T , size_t MaxLength>
typedef std::vector<T>  rti::core::bounded_sequence< T, MaxLength >::vector_type
```

The type of the underlying vector.

8.30.2.2 value_type

```
template<typename T , size_t MaxLength>
typedef vector_type::value_type  rti::core::bounded_sequence< T, MaxLength >::value_type
```

The type of the elements, `T`.

8.30.2.3 allocator_type

```
template<typename T , size_t MaxLength>
typedef vector_type::allocator_type  rti::core::bounded_sequence< T, MaxLength >::allocator_type
```

The allocator type of the underlying vector.

8.30.2.4 size_type

```
template<typename T , size_t MaxLength>
typedef vector_type::size_type  rti::core::bounded_sequence< T, MaxLength >::size_type
```

The size type.

8.30.2.5 difference_type

```
template<typename T , size_t MaxLength>
typedef vector_type::difference_type  rti::core::bounded_sequence< T, MaxLength >::difference_↩
type
```

The difference type.

8.30.2.6 reference

```
template<typename T , size_t MaxLength>
typedef vector_type::reference  rti::core::bounded_sequence< T, MaxLength >::reference
```

The reference type.

8.30.2.7 const_reference

```
template<typename T , size_t MaxLength>
typedef vector_type::const_reference  rti::core::bounded_sequence< T, MaxLength >::const_reference
```

The const reference type.

8.30.2.8 pointer

```
template<typename T , size_t MaxLength>
typedef vector_type::pointer  rti::core::bounded_sequence< T, MaxLength >::pointer
```

The pointer type.

8.30.2.9 const_pointer

```
template<typename T , size_t MaxLength>
typedef vector_type::const_pointer  rti::core::bounded_sequence< T, MaxLength >::const_pointer
```

The const pointer type.

8.30.2.10 iterator

```
template<typename T , size_t MaxLength>
typedef vector_type::iterator  rti::core::bounded_sequence< T, MaxLength >::iterator
```

The iterator type.

8.30.2.11 const_iterator

```
template<typename T , size_t MaxLength>
typedef vector_type::const_iterator  rti::core::bounded_sequence< T, MaxLength >::const_iterator
```

The const iterator type.

8.30.2.12 reverse_iterator

```
template<typename T , size_t MaxLength>
typedef vector_type::reverse_iterator  rti::core::bounded_sequence< T, MaxLength >::reverse_↩
iterator
```

The reverse iterator type.

8.30.2.13 const_reverse_iterator

```
template<typename T , size_t MaxLength>
typedef vector_type::const_reverse_iterator  rti::core::bounded_sequence< T, MaxLength >::const↩
_reverse_iterator
```

The const reverse iterator type.

8.30.3 Constructor & Destructor Documentation

8.30.3.1 bounded_sequence() [1/9]

```
template<typename T , size_t MaxLength>
rti::core::bounded_sequence< T, MaxLength >::bounded_sequence ( )  [inline]
```

Creates an empty **bounded_sequence** (p. 887).

8.30.3.2 bounded_sequence() [2/9]

```
template<typename T , size_t MaxLength>
rti::core::bounded_sequence< T, MaxLength >::bounded_sequence (
            const  bounded_sequence< T, MaxLength > & other )  [inline]
```

Copy constructor.

8.30.3.3 bounded_sequence() [3/9]

```
template<typename T , size_t MaxLength>
template<size_t M>
rti::core::bounded_sequence< T, MaxLength >::bounded_sequence (
            const  bounded_sequence< T, M > & other )  [inline], [explicit]
```

Copies a bounded sequence with a different bound `M`.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if other.size() > MaxLength

8.30.3.4 bounded_sequence() [4/9]

```
template<typename T , size_t MaxLength>
rti::core::bounded_sequence< T, MaxLength >::bounded_sequence (
             size_type count )  [inline]
```

Create a **bounded_sequence** (p. 887) with a number of default-constructed elements.

8.30.3.5 bounded_sequence() [5/9]

```
template<typename T , size_t MaxLength>
rti::core::bounded_sequence< T, MaxLength >::bounded_sequence (
             size_type count,
            const  value_type & value )  [inline]
```

Create a **bounded_sequence** (p. 887) with a number of elements.

8.30.3.6 bounded_sequence() [6/9]

```
template<typename T , size_t MaxLength>
template<typename InputIt , typename Category = typename std::iterator_traits<InputIt>::iterator↩
_category>
rti::core::bounded_sequence< T, MaxLength >::bounded_sequence (
            InputIt && first,
            InputIt && last )  [inline]
```

Create a **bounded_sequence** (p. 887) from a range of elements.

8.30.3.7 bounded_sequence() [7/9]

```
template<typename T , size_t MaxLength>
rti::core::bounded_sequence< T, MaxLength >::bounded_sequence (
            const std::vector< T > & v )  [inline]
```

Create a **bounded_sequence** (p. 887) copying the elements of a std::vector.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if v.size() > MaxLength

8.30.3.8 bounded_sequence() [8/9]

```
template<typename T , size_t MaxLength>
rti::core::bounded_sequence< T, MaxLength >::bounded_sequence (
             bounded_sequence< T, MaxLength > && other )  [inline]
```

<<***C++11***>> *(p. 353)* Move constructor

8.30.3.9 bounded_sequence() [9/9]

```
template<typename T , size_t MaxLength>
rti::core::bounded_sequence< T, MaxLength >::bounded_sequence (
            std::vector< T > && v )  [inline]
```

<<***C++11***>> *(p. 353)* Creates a **bounded_sequence** (p. 887) by moving a std::vector

8.30.4 Member Function Documentation

8.30.4.1 operator=() [1/6]

```
template<typename T , size_t MaxLength>
bounded_sequence &  rti::core::bounded_sequence< T, MaxLength >::operator= (
            const  bounded_sequence< T, MaxLength > & other )  [inline]
```

Assignment operator.

References **rti::core::bounded_sequence< T, MaxLength >::begin()**, **rti::core::bounded_sequence< T, Max↩ Length >::end()**, and **rti::core::bounded_sequence< T, MaxLength >::size()**.

8.30.4.2 operator=() [2/6]

```
template<typename T , size_t MaxLength>
template<typename U , size_t M>
bounded_sequence &  rti::core::bounded_sequence< T, MaxLength >::operator= (
            const  bounded_sequence< U, M > & other )  [inline]
```

Copies the elements of a sequence with a different bound.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if other.size() > MaxLength

References **rti::core::bounded_sequence< T, MaxLength >::begin()**, **rti::core::bounded_sequence< T, Max↩ Length >::end()**, and **rti::core::bounded_sequence< T, MaxLength >::size()**.

8.30.4.3 operator=() [3/6]

```
template<typename T , size_t MaxLength>
bounded_sequence &  rti::core::bounded_sequence< T, MaxLength >::operator= (
            const std::vector< T > & v )  [inline]
```

Copies the elements of a std::vector.

8.30.4.4 operator=() [4/6]

```
template<typename T , size_t MaxLength>
bounded_sequence &  rti::core::bounded_sequence< T, MaxLength >::operator= (
             bounded_sequence< T, MaxLength > && other )  [inline]
```

<<***C++11***>> *(p. 353)* Move-assignment operator

8.30.4.5 operator=() [5/6]

```
template<typename T , size_t MaxLength>
bounded_sequence &  rti::core::bounded_sequence< T, MaxLength >::operator= (
            std::vector< T > && v )  [inline]
```

<<***C++11***>> *(p. 353)* Moves a std::vector into this **bounded_sequence** (p. 887)

8.30.4.6 operator=() [6/6]

```
template<typename T , size_t MaxLength>
bounded_sequence &  rti::core::bounded_sequence< T, MaxLength >::operator= (
            std::initializer_list< T > l )  [inline]
```

<<***C++11***>> *(p. 353)* Copies the elements in an initializer_list

Parameters

l	The initializer list

8.30.4.7 operator[]() [1/2]

```
template<typename T , size_t MaxLength>
reference  rti::core::bounded_sequence< T, MaxLength >::operator[] (
              size_type pos )  [inline]
```

Index access.

8.30.4.8 operator[]() [2/2]

```
template<typename T , size_t MaxLength>
const_reference  rti::core::bounded_sequence< T, MaxLength >::operator[] (
              size_type pos ) const  [inline]
```

Index access.

8.30.4.9 at() [1/2]

```
template<typename T , size_t MaxLength>
reference  rti::core::bounded_sequence< T, MaxLength >::at (
              size_type pos )  [inline]
```

Index access with bounds check.

Exceptions

std::out_of_range	if pos is >= **size()** (p. 901)

8.30.4.10 at() [2/2]

```
template<typename T , size_t MaxLength>
const_reference  rti::core::bounded_sequence< T, MaxLength >::at (
              size_type pos ) const  [inline]
```

Index access with bounds check.

Exceptions

std::out_of_range	if pos is >= **size()** (p. 901)

8.30.4.11 front() [1/2]

```
template<typename T , size_t MaxLength>
reference rti::core::bounded_sequence< T, MaxLength >::front ( ) [inline]
```

Returns a reference to the first element.

8.30.4.12 front() [2/2]

```
template<typename T , size_t MaxLength>
const_reference rti::core::bounded_sequence< T, MaxLength >::front ( ) const [inline]
```

Returns a const reference to the first element.

8.30.4.13 back() [1/2]

```
template<typename T , size_t MaxLength>
reference rti::core::bounded_sequence< T, MaxLength >::back ( ) [inline]
```

Returns a reference to the last element.

8.30.4.14 back() [2/2]

```
template<typename T , size_t MaxLength>
const_reference rti::core::bounded_sequence< T, MaxLength >::back ( ) const [inline]
```

Returns a const reference to the last element.

8.30.4.15 data() [1/2]

```
template<typename T , size_t MaxLength>
pointer rti::core::bounded_sequence< T, MaxLength >::data ( ) [inline]
```

Returns a pointer to the underlying data buffer.

8.30.4.16 data() [2/2]

```
template<typename T , size_t MaxLength>
const_pointer rti::core::bounded_sequence< T, MaxLength >::data ( ) const [inline]
```

Returns a pointer to the underlying data buffer.

8.30.4.17 begin() [1/2]

```
template<typename T , size_t MaxLength>
iterator rti::core::bounded_sequence< T, MaxLength >::begin ( ) [inline]
```

Returns an iterator to the first element.

Referenced by **rti::core::bounded_sequence< T, MaxLength >::operator=()**.

8.30.4.18 begin() [2/2]

```
template<typename T , size_t MaxLength>
const_iterator rti::core::bounded_sequence< T, MaxLength >::begin ( ) const [inline]
```

Returns an const iterator to the first element.

8.30.4.19 cbegin()

```
template<typename T , size_t MaxLength>
const_iterator rti::core::bounded_sequence< T, MaxLength >::cbegin ( ) const [inline]
```

Returns an const iterator to the first element.

8.30.4.20 end() [1/2]

```
template<typename T , size_t MaxLength>
iterator rti::core::bounded_sequence< T, MaxLength >::end ( ) [inline]
```

Returns an iterator to one past the last element.

Referenced by **rti::core::bounded_sequence< T, MaxLength >::operator=()**.

8.30.4.21 end() [2/2]

```
template<typename T , size_t MaxLength>
const_iterator rti::core::bounded_sequence< T, MaxLength >::end ( ) const [inline]
```

Returns an const iterator to one past the last element.

8.30.4.22 cend()

```
template<typename T , size_t MaxLength>
const_iterator rti::core::bounded_sequence< T, MaxLength >::cend ( ) const [inline]
```

Returns a const iterator to one past the last element.

8.30.4.23 rbegin() [1/2]

```
template<typename T , size_t MaxLength>
reverse_iterator rti::core::bounded_sequence< T, MaxLength >::rbegin ( ) [inline]
```

Returns an iterator to the reverse-beginning of the sequence.

8.30.4.24 rbegin() [2/2]

```
template<typename T , size_t MaxLength>
const_reverse_iterator rti::core::bounded_sequence< T, MaxLength >::rbegin ( ) const [inline]
```

Returns a const iterator to the reverse-beginning of the sequence.

8.30.4.25 crbegin()

```
template<typename T , size_t MaxLength>
const_reverse_iterator rti::core::bounded_sequence< T, MaxLength >::crbegin ( ) const [inline]
```

Returns a const iterator to the reverse-beginning of the sequence.

8.30.4.26 rend()

```
template<typename T , size_t MaxLength>
reverse_iterator rti::core::bounded_sequence< T, MaxLength >::rend ( ) [inline]
```

Returns an iterator to the reverse-end of the sequence.

8.30.4.27 crend()

```
template<typename T , size_t MaxLength>
const_reverse_iterator rti::core::bounded_sequence< T, MaxLength >::crend ( ) const [inline]
```

Returns a const iterator to the reverse-end of the sequence.

8.30.4.28 empty()

```
template<typename T , size_t MaxLength>
bool rti::core::bounded_sequence< T, MaxLength >::empty ( ) const [inline]
```

Returns whether there are no elements.

8.30.4.29 size()

```
template<typename T , size_t MaxLength>
size_type rti::core::bounded_sequence< T, MaxLength >::size ( ) const [inline]
```

Returns the number of elements.

Referenced by **rti::core::bounded_sequence< T, MaxLength >::operator=()**.

8.30.4.30 max_size()

```
template<typename T , size_t MaxLength>
size_type  rti::core::bounded_sequence< T, MaxLength >::max_size ( ) const  [inline]
```

Returns MaxLength.

8.30.4.31 reserve()

```
template<typename T , size_t MaxLength>
void  rti::core::bounded_sequence< T, MaxLength >::reserve (
              size_type new_capacity )  [inline]
```

Pre-allocates elements up to new_capacity.

This operation reserves space for new_capacity elements and, unlike std::vector, it also default-constructs the extra elements. The **size()** (p. 901) remains the same.

8.30.4.32 capacity()

```
template<typename T , size_t MaxLength>
size_type  rti::core::bounded_sequence< T, MaxLength >::capacity ( ) const  [inline]
```

Returns the current capacity.

See also

reserve() (p. 902)

8.30.4.33 clear()

```
template<typename T , size_t MaxLength>
void  rti::core::bounded_sequence< T, MaxLength >::clear ( )  [inline]
```

Removes the elements.

This operation resizes the sequence to 0, but it doesn't destroy the elements, like std::vector would.

To destroy the elements, follow with a call to shrink_to_fit.

8.30.4.34 shrink_to_fit()

```
template<typename T , size_t MaxLength>
void  rti::core::bounded_sequence< T, MaxLength >::shrink_to_fit ( )  [inline]
```

Destroys any extra elements reserved above the current size.

Note that **resize()** (p. 904) or **clear()** (p. 902) alone won't destroy the extra elements when they reduce the size of a sequence.

For example

```
bounded_sequence<Foo, 20> s;
s.resize(10); // 10 elements
s.resize(4);  // size is now 4, but the other 6 elements are not destroyed
s.shrink_to_fit(); // size is 4, and the other 6 elements are destroyed
```

8.30.4.35 insert()

```
template<typename T , size_t MaxLength>
iterator  rti::core::bounded_sequence< T, MaxLength >::insert (
              iterator pos,
             const T & value )  [inline]
```

Inserts a new element in a position specified by an iterator.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if **size()** (p. 901) == MaxLength

8.30.4.36 erase()

```
template<typename T , size_t MaxLength>
iterator  rti::core::bounded_sequence< T, MaxLength >::erase (
              iterator pos )  [inline]
```

Erases an element specified by an iterator.

8.30.4.37 push_back() [1/2]

```
template<typename T , size_t MaxLength>
void  rti::core::bounded_sequence< T, MaxLength >::push_back (
             const T & value )  [inline]
```

Adds a new element at the end.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if **size()** (p. 901) == MaxLength

8.30.4.38 push_back() [2/2]

```
template<typename T , size_t MaxLength>
void  rti::core::bounded_sequence< T, MaxLength >::push_back (
            T && value )  [inline]
```

Adds a new element at the end by moving it.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if **size()** (p. 901) == MaxLength

8.30.4.39 pop_back()

```
template<typename T , size_t MaxLength>
void  rti::core::bounded_sequence< T, MaxLength >::pop_back ( )  [inline]
```

Removes the last element.

The element removed is not destroyed. To destroy it, follow pop_back with a call to shrink_to_fit.

8.30.4.40 resize() [1/2]

```
template<typename T , size_t MaxLength>
void  rti::core::bounded_sequence< T, MaxLength >::resize (
             size_type count )  [inline]
```

Resizes the container to contain count elements.

If count $<$ **size()** (p. 901), the sequence is reduced to the first count elements, but elements are not destroyed.

If count $>$ **size()** (p. 901), the sequence is expanded as follows: if **capacity()** (p. 902) $>=$ count, new elements are not constructed, since they already had been. If **capacity()** (p. 902) $<$ count, the extra elements are default-constructed.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if count $>$ MaxLength

8.30.4.41 resize() [2/2]

```
template<typename T , size_t MaxLength>
void  rti::core::bounded_sequence< T, MaxLength >::resize (
              size_type count,
             const T & value )  [inline]
```

Resizes the container to contain count elements.

This overload specifies the value.

8.30.4.42 swap()

```
template<typename T , size_t MaxLength>
void  rti::core::bounded_sequence< T, MaxLength >::swap (
              bounded_sequence< T, MaxLength > & other )  [inline]
```

Swap the contents of two sequences.

8.31 dds::topic::BuiltinTopicKey Class Reference

The key of the built-in topics.

```
#include <dds/topic/BuiltinTopicKey.hpp>
```

Public Member Functions

- **TBuiltinTopicKey** ()

  Creates a key whose value() are all zeros.
- std::vector< uint32_t > **value** () const

  Returns a copy of the four integers that represent the key.

Static Public Member Functions

- static D **unknown** ()

  *Unknown **BuiltinTopicKey** (p. 905).*

8.31.1 Detailed Description

The key of the built-in topics.

Each remote **dds::core::Entity** (p. 1520) to be discovered can be uniquely identified by this key. This is the key of all the built-in topic data types.

See also

dds::topic::ParticipantBuiltinTopicData (p. 1908)

dds::topic::TopicBuiltinTopicData (p. 2493)

dds::topic::PublicationBuiltinTopicData (p. 1977)

dds::topic::SubscriptionBuiltinTopicData (p. 2428)

8.31.2 Member Function Documentation

8.31.2.1 TBuiltinTopicKey()

```
dds::topic::BuiltinTopicKey::TBuiltinTopicKey ( )  [inline]
```

Creates a key whose value() are all zeros.

8.31.2.2 unknown()

```
static D dds::topic::BuiltinTopicKey::unknown ( )  [inline], [static]
```

Unknown **BuiltinTopicKey** (p. 905).

8.31.2.3 value()

```
std::vector< uint32_t > dds::topic::BuiltinTopicKey::value ( ) const
```

Returns a copy of the four integers that represent the key.

8.32 rti::core::policy::BuiltinTopicReaderResourceLimits Class Reference

<<***extension***>> *(p. 116)* Configures several resource management aspects of the built-in topic DataReaders

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **BuiltinTopicReaderResourceLimits** ()

  Creates an instance with the default settings.
- **BuiltinTopicReaderResourceLimits** & **initial_samples** (int32_t the_initial_samples)

  *<<**extension**>> (p. 116) Initial number of samples.*
- int32_t **initial_samples** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **max_samples** (int32_t the_max_samples)

  *<<**extension**>> (p. 116) Maximum number of samples.*
- int32_t **max_samples** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **initial_infos** (int32_t the_initial_infos)

  *<<**extension**>> (p. 116) Initial number of sample infos.*
- int32_t **initial_infos** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **max_infos** (int32_t the_max_infos)

  *<<**extension**>> (p. 116) Maximum number of sample infos.*
- int32_t **max_infos** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **initial_outstanding_reads** (int32_t the_initial_outstanding_reads)

  *<<**extension**>> (p. 116) The initial number of outstanding reads that have not called finish yet on the same built-in topic **dds::sub::DataReader** (p. 976).*
- int32_t **initial_outstanding_reads** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **max_outstanding_reads** (int32_t the_max_outstanding_reads)

  *<<**extension**>> (p. 116) The maximum number of outstanding reads that have not called finish yet on the same built-in topic **dds::sub::DataReader** (p. 976).*
- int32_t **max_outstanding_reads** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **max_samples_per_read** (int32_t the_max_samples_per_read)

  *<<**extension**>> (p. 116) Maximum number of samples that can be read/taken on a same built-in topic **dds::sub::DataReader** (p. 976).*
- int32_t **max_samples_per_read** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **disable_fragmentation_support** (bool the_disable_fragmentation_support)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Determines whether the built-in topic **dds::sub::DataReader** (p. 976) can receive fragmented samples.*
- bool **disable_fragmentation_support** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **max_fragmented_samples** (int32_t the_max_fragmented_samples)

  *<<**extension**>> (p. 116) The maximum number of samples for which the built-in topic **dds::sub::DataReader** (p. 976) may store fragments at a given point in time.*
- int32_t **max_fragmented_samples** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **initial_fragmented_samples** (int32_t the_initial_fragmented_samples)

  *<<**extension**>> (p. 116) The initial number of samples for which a built-in topic **dds::sub::DataReader** (p. 976) may store fragments.*

- int32_t **initial_fragmented_samples** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **max_fragmented_samples_per_remote_writer** (int32_t the_max_↩ fragmented_samples_per_remote_writer)

  <<***extension***>> (p. *116) The maximum number of samples per remote writer for which a built-in topic* ***dds::sub::***↩ ***DataReader*** (p. *976) may store fragments.*
- int32_t **max_fragmented_samples_per_remote_writer** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **max_fragments_per_sample** (int32_t the_max_fragments_per_↩ sample)

  <<***extension***>> (p. *116) Maximum number of fragments for a single sample.*
- int32_t **max_fragments_per_sample** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **dynamically_allocate_fragmented_samples** (bool the_dynamically↩ _allocate_fragmented_samples)

  <<***extension***>> (p. *116) Determines whether the built-in topic* ***dds::sub::DataReader*** (p. *976) pre-allocates storage for storing fragmented samples.*
- bool **dynamically_allocate_fragmented_samples** () const

  Getter (see setter with the same name)

8.32.1 Detailed Description

<<***extension***>> (p. *116)* Configures several resource management aspects of the built-in topic DataReaders

Defines the resources that can be used for a built-in-topic data reader.

A built-in topic data reader subscribes reliably to built-in topics containing declarations of new entities or updates to existing entities in the domain. Keys are used to differentiate among entities of the same type. RTI Connext assigns a unique key to each entity in a domain.

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

QoS:

rti::core::policy::DiscoveryConfig (p. 1271)

8.32.2 Constructor & Destructor Documentation

8.32.2.1 BuiltinTopicReaderResourceLimits()

```
rti::core::policy::BuiltinTopicReaderResourceLimits::BuiltinTopicReaderResourceLimits ( )  [inline]
```

Creates an instance with the default settings.

8.32.3 Member Function Documentation

8.32.3.1 initial_samples() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::initial_samples (
            int32_t the_initial_samples )
```

<<***extension***>> *(*p. *116)* Initial number of samples.

This should be a value between 1 and initial number of instance of the built-in-topic reader, depending on how many instances are sending data concurrently.

[default] 64

[range] [1, 1 million], <= max_samples

Categories

Immutable (p. 243), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Durability** (p. 184)

8.32.3.2 initial_samples() [2/2]

```
int32_t rti::core::policy::BuiltinTopicReaderResourceLimits::initial_samples ( ) const
```

Getter (see setter with the same name)

8.32.3.3 max_samples() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::max_↩
samples (
            int32_t the_max_samples )
```

<<***extension***>> *(p. 116)* Maximum number of samples.

This should be a value between 1 and max number of instance of the built-in-topic reader, depending on how many instances are sending data concurrently. Also, it should not be less than initial_samples.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_samples

Categories

Immutable (p. 243), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Durability** (p. 184)

8.32.3.4 max_samples() [2/2]

```
int32_t rti::core::policy::BuiltinTopicReaderResourceLimits::max_samples ( ) const
```

Getter (see setter with the same name)

8.32.3.5 initial_infos() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::initial↩
_infos (
            int32_t the_initial_infos )
```

<<***extension***>> *(p. 116)* Initial number of sample infos.

The initial number of info units that a built-in topic **dds::sub::DataReader** (p. 976) can have. Info units are used to store **dds::sub::SampleInfo** (p. 2285).

[default] 64

[range] [1, 1 million] <= max_infos

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.32.3.6 initial_infos() [2/2]

```
int32_t rti::core::policy::BuiltinTopicReaderResourceLimits::initial_infos ( ) const
```

Getter (see setter with the same name)

8.32.3.7 max_infos() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::max_infos
(
            int32_t the_max_infos )
```

<<***extension***>> *(*p. *116)* Maximum number of sample infos.

The maximum number of info units that a built-in topic **dds::sub::DataReader** (p. 976) can use to store **dds::sub::SampleInfo** (p. 2285).

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_infos

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.32.3.8 max_infos() [2/2]

```
int32_t rti::core::policy::BuiltinTopicReaderResourceLimits::max_infos ( ) const
```

Getter (see setter with the same name)

8.32.3.9 initial_outstanding_reads() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::initial
_outstanding_reads (
            int32_t the_initial_outstanding_reads )
```

<<***extension***>> *(*p. *116)* The initial number of outstanding reads that have not called finish yet on the same built-in topic **dds::sub::DataReader** (p. 976).

Must be less than or equal to max_outstanding_reads.

[default] 2

[range] [1, 1024]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Data Availability** (p. 167)

8.32.3.10 initial_outstanding_reads() [2/2]

```
int32_t rti::core::policy::BuiltinTopicReaderResourceLimits::initial_outstanding_reads ( ) const
```

Getter (see setter with the same name)

8.32.3.11 max_outstanding_reads() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::max_↩
outstanding_reads (
            int32_t the_max_outstanding_reads )
```

<<***extension***>> *(p. 116)* The maximum number of outstanding reads that have not called finish yet on the same built-in topic **dds::sub::DataReader** (p. 976).

Must be greater than or equal to initial_outstanding_reads.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1024] or **dds::core::LENGTH_UNLIMITED** (p. 438)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Data Availability** (p. 167)

8.32.3.12 max_outstanding_reads() [2/2]

```
int32_t rti::core::policy::BuiltinTopicReaderResourceLimits::max_outstanding_reads ( ) const
```

Getter (see setter with the same name)

8.32.3.13 max_samples_per_read() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::max_↩
samples_per_read (
            int32_t the_max_samples_per_read )
```

<<***extension***>> *(p. 116)* Maximum number of samples that can be read/taken on a same built-in topic **dds::sub::↩ DataReader** (p. 976).

[default] 1024

[range] [1, 65536]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Data Availability** (p. 167)

8.32.3.14 max_samples_per_read() [2/2]

```
int32_t rti::core::policy::BuiltinTopicReaderResourceLimits::max_samples_per_read ( ) const
```

Getter (see setter with the same name)

8.32.3.15 disable_fragmentation_support() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::disable↩
_fragmentation_support (
            bool the_disable_fragmentation_support )
```

[DEPRECATED] (p. **296**) <<***extension***>> (p. *116*) Determines whether the built-in topic **dds::sub::DataReader** (p. 976) can receive fragmented samples.

When fragmentation support is not needed, disabling fragmentation support will save some memory resources.

[default] false

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Fragmentation** (p. 191), **Sending Large Data** (p. 197)

8.32.3.16 disable_fragmentation_support() [2/2]

```
bool rti::core::policy::BuiltinTopicReaderResourceLimits::disable_fragmentation_support ( ) const
```

Getter (see setter with the same name)

8.32.3.17 max_fragmented_samples() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::max_↩
fragmented_samples (
            int32_t the_max_fragmented_samples )
```

<<***extension***>> *(p. 116)* The maximum number of samples for which the built-in topic **dds::sub::DataReader** (p. 976) may store fragments at a given point in time.

At any given time, a built-in topic **dds::sub::DataReader** (p. 976) may store fragments for up to `max_fragmented↩_samples` samples while waiting for the remaining fragments. These samples need not have consecutive sequence numbers and may have been sent by different built-in topic **dds::pub::DataWriter** (p. 1138) instances.

Once all fragments of a sample have been received, the sample is treated as a regular sample and becomes subject to standard QoS settings such as **rti::core::policy::BuiltinTopicReaderResourceLimits::max_samples** (p. 909).

The middleware will drop fragments if the `max_fragmented_samples` limit has been reached. For best-effort communication, the middleware will accept a fragment for a new sample, but drop the oldest fragmented sample from the same remote writer. For reliable communication, the middleware will drop fragments for any new samples until all fragments for at least one older sample from that writer have been received.

Only applies if **rti::core::policy::BuiltinTopicReaderResourceLimits::disable_fragmentation_support** (p. 913) is false.

[default] 1024

[range] [1, 1 million]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Fragmentation** (p. 191), **Sending Large Data** (p. 197)

8.32.3.18 max_fragmented_samples() [2/2]

```
int32_t rti::core::policy::BuiltinTopicReaderResourceLimits::max_fragmented_samples ( ) const
```

Getter (see setter with the same name)

8.32.3.19 initial_fragmented_samples() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::initial_fragmented_samples (
            int32_t the_initial_fragmented_samples )
```

<<***extension***>> *(*p. *116)* The initial number of samples for which a built-in topic **dds::sub::DataReader** (p. 976) may store fragments.

Only applies if **rti::core::policy::BuiltinTopicReaderResourceLimits::disable_fragmentation_support** (p. 913) is false.

[default] 4

[range] [1,1024], <= max_fragmented_samples

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Fragmentation** (p. 191), **Sending Large Data** (p. 197)

8.32.3.20 initial_fragmented_samples() [2/2]

```
int32_t rti::core::policy::BuiltinTopicReaderResourceLimits::initial_fragmented_samples ( ) const
```

Getter (see setter with the same name)

8.32.3.21 max_fragmented_samples_per_remote_writer() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::max_fragmented_samples_per_remote_writer (
            int32_t the_max_fragmented_samples_per_remote_writer )
```

<<***extension***>> *(*p. *116)* The maximum number of samples per remote writer for which a built-in topic **dds::sub::DataReader** (p. 976) may store fragments.

Logical limit so a single remote writer cannot consume all available resources.

Only applies if **rti::core::policy::BuiltinTopicReaderResourceLimits::disable_fragmentation_support** (p. 913) is false.

[default] 256

[range] [1, 1 million], <= max_fragmented_samples

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Fragmentation** (p. 191), **Sending Large Data** (p. 197)

8.32.3.22 max_fragmented_samples_per_remote_writer() [2/2]

```
int32_t rti::core::policy::BuiltinTopicReaderResourceLimits::max_fragmented_samples_per_remote_↩
writer ( ) const
```

Getter (see setter with the same name)

8.32.3.23 max_fragments_per_sample() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::max_↩
fragments_per_sample (
            int32_t the_max_fragments_per_sample )
```

<<***extension***>> *(*p. *116)* Maximum number of fragments for a single sample.

Only applies if **rti::core::policy::BuiltinTopicReaderResourceLimits::disable_fragmentation_support** (p. 913) is false.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Fragmentation** (p. 191), **Sending Large Data** (p. 197)

8.32.3.24 max_fragments_per_sample() [2/2]

```
int32_t rti::core::policy::BuiltinTopicReaderResourceLimits::max_fragments_per_sample ( ) const
```

Getter (see setter with the same name)

8.32.3.25 dynamically_allocate_fragmented_samples() [1/2]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::BuiltinTopicReaderResourceLimits::dynamically_allocate_fragmented_samples (
            bool the_dynamically_allocate_fragmented_samples )
```

<<***extension***>> *(p. 116)* Determines whether the built-in topic **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.

By default, the middleware does not allocate memory upfront, but instead allocates memory from the heap upon receiving the first fragment of a new sample. The amount of memory allocated equals the amount of memory needed to store all fragments in the sample. Once all fragments of a sample have been received, the sample is deserialized and stored in the regular receive queue. At that time, the dynamically allocated memory is freed again.

This QoS setting is useful for large, but variable-sized data types where upfront memory allocation for multiple samples based on the maximum possible sample size may be expensive. The main disadvantage of not pre-allocating memory is that one can no longer guarantee the middleware will have sufficient resources at runtime.

If `dynamically_allocate_fragmented_samples` is set to false, the middleware will allocate memory upfront for storing fragments for up to **rti::core::policy::DataReaderResourceLimits::initial_fragmented_samples** (p. 1093) samples. This memory may grow up to **rti::core::policy::DataReaderResourceLimits::max_fragmented_samples** (p. 1092) if needed.

Only applies if **rti::core::policy::DataReaderResourceLimits::disable_fragmentation_support** (p. 1092) is false.

[default] true

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Fragmentation** (p. 191), **Sending Large Data** (p. 197)

8.32.3.26 dynamically_allocate_fragmented_samples() [2/2]

```
bool rti::core::policy::BuiltinTopicReaderResourceLimits::dynamically_allocate_fragmented_samples
( ) const
```

Getter (see setter with the same name)

8.33 dds::core::BytesTopicType Class Reference

Built-in type consisting of a variable-length array of opaque bytes.

```
#include <dds/core/BuiltinTopicTypes.hpp>
```

Public Member Functions

- **BytesTopicType** ()

  Creates a sample with an empty array of bytes.
- **BytesTopicType** (const std::vector< uint8_t > &the_data)

  Creates a sample with a vector of bytes.
- **operator std::vector< uint8_t >** () const

  Automatic conversion to std::vector.
- std::vector< uint8_t > **data** () const

  Gets the bytes in a std::vector.
- void **data** (const std::vector< uint8_t > &value)

  Sets the bytes.
- uint8_t & **operator[ ]** (uint32_t index)

  Access the bytes by index.
- uint8_t **operator[ ]** (uint32_t index) const

  Access the bytes by index.
- int32_t **length** () const

  Get the number of bytes.

8.33.1 Detailed Description

Built-in type consisting of a variable-length array of opaque bytes.

See also

Built-in Types (p. 50)

8.33.2 Constructor & Destructor Documentation

8.33.2.1 BytesTopicType() [1/2]

```
dds::core::BytesTopicType::BytesTopicType ( )  [inline]
```

Creates a sample with an empty array of bytes.

8.33.2.2 BytesTopicType() [2/2]

```
dds::core::BytesTopicType::BytesTopicType (
            const std::vector< uint8_t > & the_data )  [inline]
```

Creates a sample with a vector of bytes.

Parameters

the_data	The octets

Note that this constructor is implicit so you can use a std::vector<uint8_t> wherever a **BytesTopicType** (p. 917) instance is expected

8.33.3 Member Function Documentation

8.33.3.1 operator std::vector< uint8_t >()

```
dds::core::BytesTopicType::operator std::vector< uint8_t > ( ) const  [inline]
```

Automatic conversion to std::vector.

Returns

A copy of the bytes

8.33.3.2 data() [1/2]

```
std::vector< uint8_t > dds::core::BytesTopicType::data ( ) const  [inline]
```

Gets the bytes in a std::vector.

Returns

A copy of the bytes

8.33.3.3 data() [2/2]

```
void dds::core::BytesTopicType::data (
            const std::vector< uint8_t > & value )  [inline]
```

Sets the bytes.

Parameters

value	The bytes

8.33.3.4 operator[]() [1/2]

```
uint8_t & dds::core::BytesTopicType::operator[] (
            uint32_t index )  [inline]
```

Access the bytes by index.

References **dds::core::Value< D >::delegate()**.

8.33.3.5 operator[]() [2/2]

```
uint8_t dds::core::BytesTopicType::operator[] (
            uint32_t index ) const  [inline]
```

Access the bytes by index.

References **dds::core::Value< D >::delegate()**.

8.33.3.6 length()

```
int32_t dds::core::BytesTopicType::length ( ) const  [inline]
```

Get the number of bytes.

References **dds::core::Value< D >::delegate()**.

8.34 rti::core::policy::CdrPaddingKind_def Struct Reference

<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) CdrPaddingKind

```
#include <PolicyKind.hpp>
```

Public Types

- enum **type** {
  ZERO ,
  NOT_SET ,
  AUTO }

  The underlying `enum` *type.*

8.34.1 Detailed Description

<<***extension***>> *(*p. *354)* The definition of the **dds::core::safe_enum** (p. 2265) CdrPaddingKind

The CDR padding kind determines whether or not the padding bytes will be set to zero during CDR serialization.

8.34.2 Member Enumeration Documentation

8.34.2.1 type

`enum` **rti::core::policy::CdrPaddingKind_def::type**

The underlying `enum` type.

Enumerator

ZERO	Set padding bytes to zero during CDR serialization.
NOT_SET	Don't set padding bytes to any value.
AUTO	Set the value automatically, depending on the Entity. When set on a DomainParticipant the default behavior is NOT_SET When set on a DataWriter or DataReader the behavior is to inherit the value from the DomainParticipant.

8.35 rti::core::ChannelSettings Class Reference

<<***extension***>> *(*p. *354)* Configures the properties of a channel in **rti::core::policy::MultiChannel** (p. 1749)

`#include <rti/core/PolicySettings.hpp>`

Public Member Functions

- **ChannelSettings** (const **TransportMulticastSettingsSeq** &the_multicast_settings, const std::string &the_↩ filter_expression, int32_t the_priority)

Creates an instance with the specified multicast settings, filter expression and priority.

- **ChannelSettings** & **multicast_settings** (const **TransportMulticastSettingsSeq** &the_multicast_settings)

  *<<**extension**>> (p. 116) A sequence of **rti::core::TransportMulticastSettings** (p. 2549) used to configure the multicast addresses associated with a channel.*
- **TransportMulticastSettingsSeq** **multicast_settings** () const

  Getter (see the setter with the same name)
- **ChannelSettings** & **filter_expression** (const std::string &the_filter_expression)

  *<<**extension**>> (p. 116) A logical expression used to determine the data that will be published in the channel.*
- std::string **filter_expression** () const

  Getter (see the setter with the same name)
- **ChannelSettings** & **priority** (int32_t the_priority)

  *<<**extension**>> (p. 116) Publication priority*
- int32_t **priority** () const

  Getter (see the setter with the same name)

8.35.1 Detailed Description

*<<**extension**>> (p. 354)* Configures the properties of a channel in **rti::core::policy::MultiChannel** (p. 1749)

QoS:

rti::core::policy::MultiChannel (p. 1749)

8.35.2 Constructor & Destructor Documentation

8.35.2.1 ChannelSettings()

```
rti::core::ChannelSettings::ChannelSettings (
            const  TransportMulticastSettingsSeq & the_multicast_settings,
            const std::string & the_filter_expression,
            int32_t the_priority )  [inline]
```

Creates an instance with the specified multicast settings, filter expression and priority.

See individual setters.

8.35.3 Member Function Documentation

8.35.3.1 multicast_settings() [1/2]

```
ChannelSettings & rti::core::ChannelSettings::multicast_settings (
            const  TransportMulticastSettingsSeq & the_multicast_settings )
```

<<***extension***>> *(p. 116)* A sequence of **rti::core::TransportMulticastSettings** (p. 2549) used to configure the multicast addresses associated with a channel.

The sequence cannot be empty.

The maximum number of multicast locators in a channel is limited to 16 (a locator is defined by a transport alias, a multicast address and a port). Note that this is a hard limit that cannot be increased. However, this limit can be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **rti::core::policy::Property** (p. 1969) associated with the **dds::domain::qos::DomainParticipantQos** (p. 1384).

[default] Empty sequence (invalid value)

Categories

Immutable (p. 243), **Transports** (p. 237), **Content Filtering** (p. 165)

8.35.3.2 multicast_settings() [2/2]

```
TransportMulticastSettingsSeq rti::core::ChannelSettings::multicast_settings ( ) const
```

Getter (see the setter with the same name)

8.35.3.3 filter_expression() [1/2]

```
ChannelSettings & rti::core::ChannelSettings::filter_expression (
            const std::string & the_filter_expression )  [inline]
```

<<***extension***>> *(p. 116)* A logical expression used to determine the data that will be published in the channel.

If the expression evaluates to TRUE, a sample will be published on the channel.

An empty string always evaluates the expression to TRUE.

The syntax of the expression will depend on the value of **rti::core::policy::MultiChannel::filter_name** (p. 1752)

The filter expression length (including NULL-terminated character) cannot be greater than **rti::core::policy::Domain↩ ParticipantResourceLimits::channel_filter_expression_max_length** (p. 1424).

See also

Queries and Filters Syntax (p. 84)

[default] NULL (invalid value)

Categories

Immutable (p. 243), **Content Filtering** (p. 165)

8.35.3.4 filter_expression() [2/2]

```
std::string rti::core::ChannelSettings::filter_expression ( ) const  [inline]
```

Getter (see the setter with the same name)

8.35.3.5 priority() [1/2]

```
ChannelSettings & rti::core::ChannelSettings::priority (
            int32_t the_priority )  [inline]
```

<<***extension***>> *(*p. *116)* Publication priority

A positive integer value designating the relative priority of the channel, used to determine the transmission order of pending writes.

Use of publication priorities requires the asynchronous publisher (**rti::core::policy::PublishModeKind_def::↩ ASYNCHRONOUS** (p. 2019)) with **rti::pub::FlowControllerProperty::scheduling_policy** (p. 1579) set to Flow↩ ControllerSchedulingPolicy_def::HIGHEST_PRIORITY_FIRST.

Larger numbers have higher priority.

If the publication priority of the channel is any value other than PUBLICATION_PRIORITY_UNDEFINED, then the channel's priority will take precedence over the data writer's priority.

If the publication priority of the channel is set to PUBLICATION_PRIORITY_UNDEFINED, then the channel's priority will be set to the value of the data writer's priority.

If the publicaton priority of both the data writer and the channel are PUBLICATION_PRIORITY_UNDEFINED, the channel will be assigned the lowest priority value.

If the publication priority of the channel is PUBLICATION_PRIORITY_AUTOMATIC, then the channel will be assigned the priority of the largest publication priority of all samples in the channel. The publication priority of each sample can be set in the **rti::pub::WriteParams** (p. 2642) of the **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179) function.

[default] PUBLICATION_PRIORITY_UNDEFINED

Categories

Immutable (p. 243), **Transports** (p. 237), **Data Availability** (p. 167)

8.35.3.6 priority() [2/2]

```
int32_t rti::core::ChannelSettings::priority ( ) const  [inline]
```

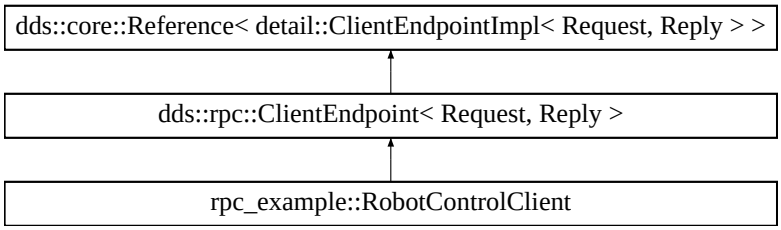
Getter (see the setter with the same name)

8.36 dds::rpc::ClientEndpoint< Request, Reply > Class Template Reference

<<***reference-type***>> *(p. 351)* Manages the DDS entities required to make remote function calls.

`#include <dds/rpc/ClientEndpoint.hpp>`

Inheritance diagram for dds::rpc::ClientEndpoint< Request, Reply >:



Public Types

- using **RequestType** = Request

  The type used to make function calls.
- using **ReplyType** = Reply

  The type used to receive the results of function calls.

Public Member Functions

- **ClientEndpoint** (const **ClientParams** &params)

  *Creates a new **ClientEndpoint** (p. 925).*
- void **close** ()

  Destroyes the underlying requester and other resources.
- bool **closed** () const

  Whether this instance has been closed already.
- bool **wait_for_service** (const **dds::core::Duration** &max_wait= **dds::core::Duration::infinite**())

  Waits for one or more services to be discovered. This function takes advantage of enhanced discovery, making sure both the Client and the Service discover each other.
- **dds::pub::DataWriter**< **RequestType** > **request_datawriter** () const

  Accesses the underlying DataWriter that sends the requests (function calls)
- **dds::sub::DataReader**< **ReplyType** > **reply_datareader** () const

  Accesses the underlying DataReader that receives the replies (return values from the function calls)

8.36.1 Detailed Description

template<**typename Request, typename Reply**>
class dds::rpc::ClientEndpoint< **Request, Reply** >

<<***reference-type***>> *(p. 351)* Manages the DDS entities required to make remote function calls.

This class is always the base of a concrete client, such as **rpc_example::RobotControlClient** (p. 2217).

8.36.2 Member Typedef Documentation

8.36.2.1 RequestType

```
template<typename Request , typename Reply >
using  dds::rpc::ClientEndpoint< Request, Reply >::RequestType = Request
```

The type used to make function calls.

8.36.2.2 ReplyType

```
template<typename Request , typename Reply >
using  dds::rpc::ClientEndpoint< Request, Reply >::ReplyType = Reply
```

The type used to receive the results of function calls.

8.36.3 Constructor & Destructor Documentation

8.36.3.1 ClientEndpoint()

```
template<typename Request , typename Reply >
dds::rpc::ClientEndpoint< Request, Reply >::ClientEndpoint (
            const  ClientParams & params )  [inline], [explicit]
```

Creates a new **ClientEndpoint** (p. 925).

8.36.4 Member Function Documentation

8.36.4.1 close()

```
template<typename Request , typename Reply >
void  dds::rpc::ClientEndpoint< Request, Reply >::close ( )  [inline]
```

Destroyes the underlying requester and other resources.

Any operation after **close()** (p. 926) throws **dds::core::AlreadyClosedError** (p. 806)

8.36.4.2 closed()

```
template<typename Request , typename Reply >
bool  dds::rpc::ClientEndpoint< Request, Reply >::closed ( ) const  [inline]
```

Whether this instance has been closed already.

8.36.4.3 wait_for_service()

```
template<typename Request , typename Reply >
bool  dds::rpc::ClientEndpoint< Request, Reply >::wait_for_service (
            const  dds::core::Duration & max_wait =  dds::core::Duration::infinite() )  [inline]
```

Waits for one or more services to be discovered. This function takes advantage of enhanced discovery, making sure both the Client and the Service discover each other.

Parameters

max_wait	The maximum time to wait. Default: **dds::core::Duration::infinite()** (p. 1451) (unlimited period to wait).

Returns

True if a Service is discovered. False if a timeout occurs.

Exceptions

dds::core::AlreadyClosedError *(p. 806)*	if the Client is closed while waiting.

8.36.4.4 request_datawriter()

```
template<typename Request , typename Reply >
dds::pub::DataWriter<  RequestType >  dds::rpc::ClientEndpoint< Request, Reply >::request_↩
datawriter ( ) const  [inline]
```

Accesses the underlying DataWriter that sends the requests (function calls)

Returns

The DataWriter

8.36.4.5 reply_datareader()

```
template<typename Request , typename Reply >
dds::sub::DataReader< ReplyType > dds::rpc::ClientEndpoint< Request, Reply >::reply_datareader
( ) const [inline]
```

Accesses the underlying DataReader that receives the replies (return values from the function calls)

Returns

The DataReader

8.37 dds::rpc::ClientParams Class Reference

<<***value-type***>> *(p. 350)* The parameters used to configure a **ClientEndpoint** (p. 925)

`#include <dds/rpc/ClientEndpoint.hpp>`

Inherits rti::request::detail::EntityParamsWithSetters< ActualEntity >.

Public Member Functions

- **ClientParams** (**dds::domain::DomainParticipant** participant)
  Construct a new Client Params object.
- **ClientParams** & **function_call_max_wait** (const **dds::core::Duration** &max_wait)
  Specifies the maximum wait time for all the remote calls.
- **dds::core::Duration function_call_max_wait** () const
  Returns the maximum wait time.

8.37.1 Detailed Description

<<***value-type***>> *(p. 350)* The parameters used to configure a **ClientEndpoint** (p. 925)

Since clients use a **rti::request::Requester** (p. 2187) to communicate with a service, the **ClientParams** (p. 928) type contains the same parameters as an **rti::request::RequesterParams** (p. 2199), plus function_call_max_wait.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 ClientParams()

```
dds::rpc::ClientParams::ClientParams (
            dds::domain::DomainParticipant participant ) [inline]
```

Construct a new Client Params object.

Parameters

participant	The DomainParticipant is a required parameter

8.37.3 Member Function Documentation

8.37.3.1 function_call_max_wait() [1/2]

```
ClientParams & dds::rpc::ClientParams::function_call_max_wait (
            const  dds::core::Duration & max_wait )  [inline]
```

Specifies the maximum wait time for all the remote calls.

Parameters

max_wait	The maximum wait time for all remote calls made by a client

Returns

This object

8.37.3.2 function_call_max_wait() [2/2]

```
dds::core::Duration dds::rpc::ClientParams::function_call_max_wait ( ) const  [inline]
```

Returns the maximum wait time.

8.38 dds::sub::CoherentAccess Class Reference

<<***value-type***>> *(p. 350)* Controls whether RTI Connext will preserve the groupings of changes made by the publishing application by means of begin_coherent_changes and end_coherent_changes.

```
#include "dds/sub/CoherentAccess.hpp"
```

Public Member Functions

- **CoherentAccess** (const **dds::sub::Subscriber** &sub)

  *Creating a **CoherentAccess*** (p. *929*) *object indicates that the application is about to access the data samples in any of the **DataReader*** (p. *976*) *objects attached to the provided **Subscriber*** (p. *2410*).

- void **end** ()

  *Explicitly indicate that the application has finished accessing the data samples in **DataReader*** (p. *976*) *objects managed by the **Subscriber*** (p. *2410*).

- ∼**CoherentAccess** ()

  The destructor implicitly ends coherent access.

8.38.1 Detailed Description

<<***value-type***>> (p. *350*) Controls whether RTI Connext will preserve the groupings of changes made by the publishing application by means of begin_coherent_changes and end_coherent_changes.

8.38.2 Constructor & Destructor Documentation

8.38.2.1 CoherentAccess()

```
dds::sub::CoherentAccess::CoherentAccess (
            const  dds::sub::Subscriber & sub )  [inline], [explicit]
```

Creating a **CoherentAccess** (p. 929) object indicates that the application is about to access the data samples in any of the **DataReader** (p. 976) objects attached to the provided **Subscriber** (p. 2410).

If the **dds::core::policy::Presentation::access_scope** (p. 1947) of the **dds::sub::Subscriber** (p. 2410) is **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951) or **dds::core::policy::PresentationAccessScopeKind_def::HIGHEST_OFFERED** (p. 1951) and **dds::core::policy::Presentation::ordered_access** (p. 1948) is true, the application is required to use this operation to access the samples in order across DataWriters of the same group (**dds::pub::Publisher** (p. 1993) with **dds::core::policy::Presentation::access_scope** (p. 1947) set to **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951)).

In the above case, this operation must be called prior to calling any of the sample-accessing operations, or **dds::sub::find** (p. 672)

Once the application has finished accessing the data samples, it must call **dds::sub::CoherentAccess::end()** (p. 931) or let this object be destroyed.

The application is not required to call **dds::sub::CoherentAccess::CoherentAccess()** (p. 930) / **dds::sub::CoherentAccess::end()** (p. 931) to access the samples in order if the **PRESENTATION** (p. 540) policy in the **dds::pub::Publisher** (p. 1993) has **dds::core::policy::Presentation::access_scope** (p. 1947) set to something other than **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951). In this case, calling **dds::sub::CoherentAccess::CoherentAccess()** (p. 930) / **dds::sub::CoherentAccess::end()** (p. 931) is not considered an error and has no effect.

Calls to **dds::sub::CoherentAccess::CoherentAccess()** (p. 930) / **dds::sub::CoherentAccess::end()** (p. 931) may be nested and must be balanced.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869).

See also

Access to data samples (p. 62)

dds::sub::find (p. 672)

PRESENTATION (p. 540)

8.38.2.2 ~CoherentAccess()

```
dds::sub::CoherentAccess::~CoherentAccess ( )  [inline]
```

The destructor implicitly ends coherent access.

See also

end() (p. 931)

8.38.3 Member Function Documentation

8.38.3.1 end()

```
void dds::sub::CoherentAccess::end ( )  [inline]
```

Explicitly indicate that the application has finished accessing the data samples in **DataReader** (p. 976) objects managed by the **Subscriber** (p. 2410).

This operation must be used to close a corresponding begin_access().

This call must close a previous call to **dds::sub::CoherentAccess::CoherentAccess()** (p. 930)(), otherwise the operation will fail with the error **dds::core::PreconditionNotMetError** (p. 1941).

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::PreconditionNotMetError** (p. 1941) or **dds::core::NotEnabledError** (p. 1869).

8.39 dds::pub::CoherentSet Class Reference

<<***value-type***>> *(*p. *350)* A publishing application can request that a set of DDS data-sample changes be propagated in such a way that they are interpreted at the receivers' side as a cohesive set of modifications.

```
#include "dds/pub/CoherentSet.hpp"
```

Public Member Functions

- **CoherentSet** (const **dds::pub::Publisher** &pub)

  Creating a ***CoherentSet*** *(*p. *932) object indicates that the application will begin a coherent set of modifications using* ***dds::pub::DataWriter*** *(*p. *1138) objects attached to the* ***dds::pub::Publisher*** *(*p. *1993).*
- void **end** ()

  Explicitly terminate a coherent set initiated by the ***CoherentSet*** *(*p. *932) constructor.*
- ∼**CoherentSet** ()

  Implicitly terminate a coherent set.

8.39.1 Detailed Description

<<***value-type***>> *(*p. *350)* A publishing application can request that a set of DDS data-sample changes be propagated in such a way that they are interpreted at the receivers' side as a cohesive set of modifications.

In this case, the receiver will only be able to access the data after all the modifications in the set are available at the subscribing end.

8.39.2 Constructor & Destructor Documentation

8.39.2.1 CoherentSet()

```
dds::pub::CoherentSet::CoherentSet (
            const  dds::pub::Publisher & pub )  [inline], [explicit]
```

Creating a **CoherentSet** (p. 932) object indicates that the application will begin a coherent set of modifications using **dds::pub::DataWriter** (p. 1138) objects attached to the **dds::pub::Publisher** (p. 1993).

A 'coherent set' is a set of modifications that must be propagated in such a way that they are interpreted at the receiver's side as a consistent set of modifications; that is, the receiver will only be able to access the data after all the modifications in the set are available at the receiver end.

A connectivity change may occur in the middle of a set of coherent changes; for example, the set of partitions used by the **dds::pub::Publisher** (p. 1993) or one of its subscribers (**dds::sub::Subscriber** (p. 2410)) may change, a late-joining **dds::sub::DataReader** (p. 976) may appear on the network, or a communication failure may occur. In the event that such a change prevents an entity from receiving the entire set of coherent changes, that entity must behave as if it had received none of the set.

These calls can be nested. In that case, the coherent set terminates only with the last call to **dds::pub::CoherentSet::end()** (p. 933). **Publisher** (p. 1993)'s samples (samples published by any of the DataWriters within the **Publisher** (p. 1993)) that are not published within a begin_coherent_changes/end_coherent_changes block will not be provided to the DataReaders as a set.

The support for coherent changes enables a publishing application to change the value of several data-instances that could belong to the same or different topics and have those changes be seen *atomically* by the readers. This is useful in cases where the values are inter-related (for example, if there are two data-instances representing the altitude and velocity vector of the same aircraft and both are changed, it may be useful to communicate those values in a way the reader can see both together; otherwise, it may, e.g., erroneously interpret that the aircraft is on a collision course).

Note

Coherent sets don't apply to Topic Queries. If a **rti::sub::TopicQuery** (p. 2516) selects only a subset of samples that was published as a coherent set, the subscribing application will receive them regardless of their membership to the coherent set.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869).

See also

dds::core::policy::Presentation (p. 1942)

Parameters

pub	The **Publisher** (p. 1993)

8.39.2.2 ~CoherentSet()

```
dds::pub::CoherentSet::~CoherentSet ( )  [inline]
```

Implicitly terminate a coherent set.

See also

end() (p. 933)

8.39.3 Member Function Documentation

8.39.3.1 end()

```
void dds::pub::CoherentSet::end ( )  [inline]
```

Explicitly terminate a coherent set initiated by the **CoherentSet** (p. 932) constructor.

Precondition

If there is no matching call to **dds::pub::CoherentSet::CoherentSet** (p. 932) the operation will fail with **dds::core::PreconditionNotMetError** (p. 1941).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::PreconditionNotMetError** (p. 1941) or **dds::core::NotEnabledError** (p. 1869).

8.40 rti::core::CoherentSetInfo Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* A CoherentSampleInfo provides information about the coherent set associated with a sample.

```
#include "rti/core/CoherentSetInfo.hpp"
```

Public Member Functions

- **CoherentSetInfo** ()

  Create a default CoherentSampleInfo object.
- **CoherentSetInfo** (const **Guid** &the_group_guid, const **SequenceNumber** &the_coherent_set_sequence_number, const **SequenceNumber** &the_group_coherent_set_sequence_number, bool the_incomplete_coherent_set)

  Creates CoherentSampleInfo object with the specified parameters.
- const **Guid** & **group_guid** () const

  *Gets the coherent set group **Guid** (p. 1596).*
- **Guid** & **group_guid** ()

  *Gets the coherent set group **Guid** (p. 1596).*
- **CoherentSetInfo** & **group_guid** (const **Guid** &value)

  *Sets the coherent set group **Guid** (p. 1596).*
- const **SequenceNumber** & **coherent_set_sequence_number** () const

  Gets the coherent set sequence number that identifies a sample as part of a DataWriter coherent set.
- **SequenceNumber** & **coherent_set_sequence_number** ()

  Gets the coherent set sequence number that identifies a sample as part of a DataWriter coherent set.
- **CoherentSetInfo** & **coherent_set_sequence_number** (const **SequenceNumber** &value)

  Sets the coherent set sequence number that identifies a sample as part of a DataWriter coherent set.
- const **SequenceNumber** & **group_coherent_set_sequence_number** () const

Gets the group coherent set sequence number that identifies a sample as part of a group coherent set.

- **SequenceNumber** & **group_coherent_set_sequence_number** ()

  Gets the group coherent set sequence number that identifies a sample as part of a group coherent set.
- **CoherentSetInfo** & **group_coherent_set_sequence_number** (const **SequenceNumber** &value)

  Gets the group coherent set sequence number that identifies a sample as part of a group coherent set.
- bool **incomplete_coherent_set** () const

  Indicates if a sample is part of an incomplete coherent set.
- **CoherentSetInfo** & **incomplete_coherent_set** (bool value)

  Sets the incomplete coherent set status.

Static Public Member Functions

- static **CoherentSetInfo** **unknown** ()

  An invalid or unknown coherent set info.

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator**<< (std::ostream &out, const **CoherentSetInfo** &csi)

  *Prints a **CoherentSetInfo** (p. 934) to an output stream.*

8.40.1 Detailed Description

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* A CoherentSampleInfo provides information about the coherent set associated with a sample.

8.40.2 Constructor & Destructor Documentation

8.40.2.1 CoherentSetInfo() [1/2]

```
rti::core::CoherentSetInfo::CoherentSetInfo ( )  [inline]
```

Create a default CoherentSampleInfo object.

8.40.2.2 CoherentSetInfo() [2/2]

```
rti::core::CoherentSetInfo::CoherentSetInfo (
            const  Guid & the_group_guid,
            const  SequenceNumber & the_coherent_set_sequence_number,
            const  SequenceNumber & the_group_coherent_set_sequence_number,
            bool the_incomplete_coherent_set )  [inline]
```

Creates CoherentSampleInfo object with the specified parameters.

8.40.3 Member Function Documentation

8.40.3.1 group_guid() [1/3]

```
const  Guid & rti::core::CoherentSetInfo::group_guid ( ) const  [inline]
```

Gets the coherent set group **Guid** (p. 1596).

This GUID identifies the **dds::pub::DataWriter** (p. 1138) or the group of DataWriters publishing the coherent set, depending on the value of **dds::core::policy::Presentation::access_scope** (p. 1947) in the **dds::sub::Subscriber** (p. 2410).

8.40.3.2 group_guid() [2/3]

```
Guid & rti::core::CoherentSetInfo::group_guid ( )  [inline]
```

Gets the coherent set group **Guid** (p. 1596).

This GUID identifies the **dds::pub::DataWriter** (p. 1138) or the group of DataWriters publishing the coherent set, depending on the value of **dds::core::policy::Presentation::access_scope** (p. 1947) in the **dds::sub::Subscriber** (p. 2410).

8.40.3.3 group_guid() [3/3]

```
CoherentSetInfo & rti::core::CoherentSetInfo::group_guid (
            const  Guid & value )  [inline]
```

Sets the coherent set group **Guid** (p. 1596).

This GUID identifies the **dds::pub::DataWriter** (p. 1138) or the group of DataWriters publishing the coherent set, depending on the value of **dds::core::policy::Presentation::access_scope** (p. 1947) in the **dds::sub::Subscriber** (p. 2410).

8.40.3.4 coherent_set_sequence_number() [1/3]

```
const SequenceNumber & rti::core::CoherentSetInfo::coherent_set_sequence_number ( ) const [inline]
```

Gets the coherent set sequence number that identifies a sample as part of a DataWriter coherent set.

When **dds::core::policy::Presentation::access_scope** (p. 1947) in the **dds::sub::Subscriber** (p. 2410) is set to **dds::core::policy::PresentationAccessScopeKind_def::INSTANCE** (p. 1951) or **dds::core::policy::↩ PresentationAccessScopeKind_def::TOPIC** (p. 1951), the coherent set associated with a sample is identified by the pair (group_guid, coherent_set_sequence_number).

8.40.3.5 coherent_set_sequence_number() [2/3]

```
SequenceNumber & rti::core::CoherentSetInfo::coherent_set_sequence_number ( ) [inline]
```

Gets the coherent set sequence number that identifies a sample as part of a DataWriter coherent set.

When **dds::core::policy::Presentation::access_scope** (p. 1947) in the **dds::sub::Subscriber** (p. 2410) is set to **dds::core::policy::PresentationAccessScopeKind_def::INSTANCE** (p. 1951) or **dds::core::policy::↩ PresentationAccessScopeKind_def::TOPIC** (p. 1951), the coherent set associated with a sample is identified by the pair (group_guid, coherent_set_sequence_number).

8.40.3.6 coherent_set_sequence_number() [3/3]

```
CoherentSetInfo & rti::core::CoherentSetInfo::coherent_set_sequence_number (
            const SequenceNumber & value ) [inline]
```

Sets the coherent set sequence number that identifies a sample as part of a DataWriter coherent set.

This GUID identifies the **dds::pub::DataWriter** (p. 1138) or the group of DataWriters publishing the coherent set, depending on the value of **dds::core::policy::Presentation::access_scope** (p. 1947) in the **dds::sub::Subscriber** (p. 2410).

8.40.3.7 group_coherent_set_sequence_number() [1/3]

```
const SequenceNumber & rti::core::CoherentSetInfo::group_coherent_set_sequence_number ( ) const
[inline]
```

Gets the group coherent set sequence number that identifies a sample as part of a group coherent set.

When **dds::core::policy::Presentation::access_scope** (p. 1947) in the **dds::sub::Subscriber** (p. 2410) is set to **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951), the coherent set associated with a sample is identified by the pair (group_guid, group_coherent_set_sequence_number).

8.40.3.8 group_coherent_set_sequence_number() [2/3]

```
SequenceNumber & rti::core::CoherentSetInfo::group_coherent_set_sequence_number ( )  [inline]
```

Gets the group coherent set sequence number that identifies a sample as part of a group coherent set.

When **dds::core::policy::Presentation::access_scope** (p. 1947) in the **dds::sub::Subscriber** (p. 2410) is set to **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951), the coherent set associated with a sample is identified by the pair (group_guid, group_coherent_set_sequence_number).

8.40.3.9 group_coherent_set_sequence_number() [3/3]

```
CoherentSetInfo & rti::core::CoherentSetInfo::group_coherent_set_sequence_number (
            const  SequenceNumber & value )  [inline]
```

Gets the group coherent set sequence number that identifies a sample as part of a group coherent set.

This GUID identifies the **dds::pub::DataWriter** (p. 1138) or the group of DataWriters publishing the coherent set, depending on the value of **dds::core::policy::Presentation::access_scope** (p. 1947) in the **dds::sub::Subscriber** (p. 2410).

8.40.3.10 incomplete_coherent_set() [1/2]

```
bool rti::core::CoherentSetInfo::incomplete_coherent_set ( ) const  [inline]
```

Indicates if a sample is part of an incomplete coherent set.

8.40.3.11 incomplete_coherent_set() [2/2]

```
CoherentSetInfo & rti::core::CoherentSetInfo::incomplete_coherent_set (
            bool value )  [inline]
```

Sets the incomplete coherent set status.

8.40.3.12 unknown()

```
static  CoherentSetInfo rti::core::CoherentSetInfo::unknown ( )  [inline], [static]
```

An invalid or unknown coherent set info.

8.40.4 Friends And Related Function Documentation

8.40.4.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const  CoherentSetInfo & csi )  [related]
```

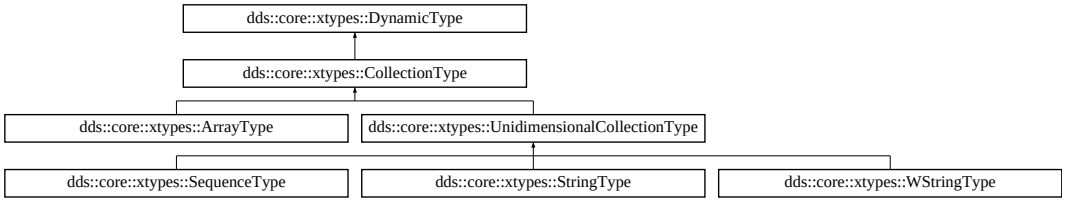
Prints a **CoherentSetInfo** (p. 934) to an output stream.

8.41 dds::core::xtypes::CollectionType Class Reference

<<***value-type***>> *(p. 350)* The base class of all collection types

`#include <dds/core/xtypes/CollectionTypes.hpp>`

Inheritance diagram for dds::core::xtypes::CollectionType:



Public Member Functions

- const **dds::core::xtypes::DynamicType** & **content_type** () const
  Gets the type of the elements of this collection.

Additional Inherited Members

8.41.1 Detailed Description

<<***value-type***>> *(p. 350)* The base class of all collection types

8.41.2 Member Function Documentation

8.41.2.1 content_type()

```
const  dds::core::xtypes::DynamicType & dds::core::xtypes::CollectionType::content_type ( ) const
```

Gets the type of the elements of this collection.

8.42 rti::core::CompressionIdMask Class Reference

<<***extension***>> *(*p. *354)* Mask that specifies which built-in compression method to used.

```
#include <rti/core/PolicySettings.hpp>
```

Public Types

- typedef std::bitset< 16 > **MaskType**
  The base type, std::bitset.

Public Member Functions

- **CompressionIdMask** (uint64_t mask)
  Creates a mask from the bits in an integer.
- **CompressionIdMask** (const **MaskType** &mask)
  Creates a mask from a std::bitset.

Static Public Member Functions

- static const **CompressionIdMask** **all** ()
  All bits are set.
- static const **CompressionIdMask** **none** ()
  No bits are set.
- static const **CompressionIdMask** **default_publication** ()
  Default mask value for publication.
- static const **CompressionIdMask** **default_subscription** ()
  Default mask value for subscription.
- static const **CompressionIdMask** **zlib** ()
  Selects the built-in ZLIB compression algorithm.
- static const **CompressionIdMask** **bzip2** ()
  Selects the built-in BZIP2 compression algorithm.
- static const **CompressionIdMask** **lz4** ()
  Selects the built-in LZ4 compression algorithm.

8.42.1 Detailed Description

<<***extension***>> *(p. 354)* Mask that specifies which built-in compression method to used.

8.42.2 Member Typedef Documentation

8.42.2.1 MaskType

```
typedef std::bitset<16>  rti::core::CompressionIdMask::MaskType
```

The base type, std::bitset.

8.42.3 Constructor & Destructor Documentation

8.42.3.1 CompressionIdMask() [1/2]

```
rti::core::CompressionIdMask::CompressionIdMask (
            uint64_t mask )  [inline], [explicit]
```

Creates a mask from the bits in an integer.

8.42.3.2 CompressionIdMask() [2/2]

```
rti::core::CompressionIdMask::CompressionIdMask (
            const  MaskType & mask )  [inline]
```

Creates a mask from a std::bitset.

8.42.4 Member Function Documentation

8.42.4.1 all()

```
static const CompressionIdMask rti::core::CompressionIdMask::all ( ) [inline], [static]
```

All bits are set.

8.42.4.2 none()

```
static const CompressionIdMask rti::core::CompressionIdMask::none ( ) [inline], [static]
```

No bits are set.

8.42.4.3 default_publication()

```
static const CompressionIdMask rti::core::CompressionIdMask::default_publication ( ) [inline],
[static]
```

Default mask value for publication.

8.42.4.4 default_subscription()

```
static const CompressionIdMask rti::core::CompressionIdMask::default_subscription ( ) [inline],
[static]
```

Default mask value for subscription.

8.42.4.5 zlib()

```
static const CompressionIdMask rti::core::CompressionIdMask::zlib ( ) [inline], [static]
```

Selects the built-in ZLIB compression algorithm.

8.42.4.6 bzip2()

```
static const CompressionIdMask rti::core::CompressionIdMask::bzip2 ( ) [inline], [static]
```

Selects the built-in BZIP2 compression algorithm.

8.42.4.7 lz4()

```
static const CompressionIdMask rti::core::CompressionIdMask::lz4 ( ) [inline], [static]
```

Selects the built-in LZ4 compression algorithm.

8.43 rti::core::CompressionSettings Class Reference

<<***extension***>> *(*p. *354)* Compression Settings

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **CompressionSettings** ()

  Creates an instance with default, compression_ids, writer_compression_level and writer_compression_threshold.
- **CompressionSettings** (const **rti::core::CompressionIdMask compression_ids**)

  Creates an instance with the given compression_ids.
- **CompressionSettings** (const **rti::core::CompressionIdMask compression_ids**, const uint32_t **writer_compression_level**, const int32_t **writer_compression_threshold**)

  Creates an instance with the given compression_ids, writer_compression_level and writer_compression_threshold.
- **rti::core::CompressionIdMask compression_ids** () const

  Getter for compression_ids.
- **CompressionSettings compression_ids** (**rti::core::CompressionIdMask** compression_ids)

  Setter for compression_ids.
- uint32_t **writer_compression_level** () const

  Getter for writer_compression_level.
- **CompressionSettings writer_compression_level** (uint32_t writer_compression_level)

  Setter for writer_compression_level.
- int32_t **writer_compression_threshold** () const

  Getter for writer_compression_threshold.
- **CompressionSettings writer_compression_threshold** (int32_t writer_compression_threshold)

  Setter for writer_compression_threshold.

Static Public Member Functions

- static uint32_t **compression_level_default** ()

  Return the default value for the compression level.
- static uint32_t **compression_level_best_compression** ()

  Get the compression level that can be used to achieve the best compression ratio.
- static uint32_t **compression_level_best_speed** ()

  Get the compression level that can be used to achieve the best compression speed.

8.43.1 Detailed Description

<<***extension***>> *(*p. *354)* Compression Settings

QoS:

dds::core::policy::DataRepresentation (p. 1113)

8.43.2 Constructor & Destructor Documentation

8.43.2.1 CompressionSettings() [1/3]

```
rti::core::CompressionSettings::CompressionSettings ( )  [inline]
```

Creates an instance with default, compression_ids, writer_compression_level and writer_compression_threshold.

8.43.2.2 CompressionSettings() [2/3]

```
rti::core::CompressionSettings::CompressionSettings (
            const  rti::core::CompressionIdMask compression_ids )  [inline], [explicit]
```

Creates an instance with the given compression_ids.

8.43.2.3 CompressionSettings() [3/3]

```
rti::core::CompressionSettings::CompressionSettings (
            const  rti::core::CompressionIdMask compression_ids,
            const uint32_t writer_compression_level,
            const int32_t writer_compression_threshold )  [inline]
```

Creates an instance with the given compression_ids, writer_compression_level and writer_compression_threshold.

References **compression_ids()**, **compression_level_default()**, **writer_compression_level()**, and **writer_↩ compression_threshold()**.

8.43.3 Member Function Documentation

8.43.3.1 compression_level_default()

```
static uint32_t rti::core::CompressionSettings::compression_level_default ( )  [inline], [static]
```

Return the default value for the compression level.

Referenced by **CompressionSettings()**.

8.43.3.2 compression_level_best_compression()

```
static uint32_t rti::core::CompressionSettings::compression_level_best_compression ( )  [inline],
[static]
```

Get the compression level that can be used to achieve the best compression ratio.

8.43.3.3 compression_level_best_speed()

```
static uint32_t rti::core::CompressionSettings::compression_level_best_speed ( )  [inline], [static]
```

Get the compression level that can be used to achieve the best compression speed.

8.43.3.4 compression_ids() [1/2]

```
rti::core::CompressionIdMask rti::core::CompressionSettings::compression_ids ( ) const  [inline]
```

Getter for compression_ids.

Referenced by **CompressionSettings()**.

8.43.3.5 compression_ids() [2/2]

```
CompressionSettings rti::core::CompressionSettings::compression_ids (
            rti::core::CompressionIdMask compression_ids )  [inline]
```

Setter for compression_ids.

A bitmap that represents the compression algorithm IDs (**rti::core::CompressionIdMask** (p. 940)) that are supported by the endpoint. The **dds::pub::DataWriter** (p. 1138) creation will fail if more than one algorithm is provided.

If a **dds::pub::DataWriter** (p. 1138) inherits multiple compression IDs from a **dds::topic::Topic** (p. 2474), only the least significant bit enabled will be inherited. This forces the following order of preference: **rti::core::CompressionIdMask↩ ::zlib()** (p. 942), **rti::core::CompressionIdMask::bzip2()** (p. 942), **rti::core::CompressionIdMask::lz4()** (p. 943).

Interactions with Security and Batching: Currently, the only algorithm that is supported when compression and batching are enabled on the same **dds::pub::DataWriter** (p. 1138) is **rti::core::CompressionIdMask::zlib()** (p. 942).

The combination of compression, batching, and data protection is supported. First, compression is applied to the entire batch. Then, data protection is applied to the compressed batch.

Note: When **rti::core::policy::DataWriterProtocol::serialize_key_with_dispose** (p. 1215) is enabled and a dispose message is sent, the serialized key is not compressed.

[default] For **dds::topic::Topic** (p. 2474), **dds::pub::DataWriter** (p. 1138) a **rti::core::CompressionIdMask** (p. 940) mask set to **rti::core::CompressionIdMask::none()** (p. 942)

[default] For **dds::sub::DataReader** (p. 976) a **rti::core::CompressionIdMask** (p. 940) mask set to **rti::core::↩ CompressionIdMask::all()** (p. 941).

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Sending Large Data** (p. 197)

8.43.3.6 writer_compression_level() [1/2]

```
uint32_t rti::core::CompressionSettings::writer_compression_level ( ) const  [inline]
```

Getter for writer_compression_level.

Referenced by **CompressionSettings()**.

8.43.3.7 writer_compression_level() [2/2]

```
CompressionSettings rti::core::CompressionSettings::writer_compression_level (
            uint32_t writer_compression_level )  [inline]
```

Setter for writer_compression_level.

Compression algorithms typically allow you to choose a level with which to compress the data. Each level has trade-offs between the resulting compression ratio and the speed of compression.

[range] [0, 10]

The value 1 represents the fastest compression time and the lowest compression ratio. The value 10 represents the slowest compression time but the highest compression ratio.

A value of 0 disables compression.

[default] rti::core::CompressionSettings::compression_level_best_compression() (p. 945)

Note

Only available for a **dds::pub::DataWriter** (p. 1138) and **dds::topic::Topic** (p. 2474).

Categories

Immutable (p. 243), **Sending Large Data** (p. 197)

8.43.3.8 writer_compression_threshold() [1/2]

```
int32_t rti::core::CompressionSettings::writer_compression_threshold ( ) const  [inline]
```

Getter for writer_compression_threshold.

Referenced by **CompressionSettings()**.

8.43.3.9 writer_compression_threshold() [2/2]

```
CompressionSettings rti::core::CompressionSettings::writer_compression_threshold (
            int32_t writer_compression_threshold )  [inline]
```

Setter for writer_compression_threshold.

Any sample with a serialized size greater than or equal to the threshold will be eligible to be compressed. All samples with an eligible serialized size will be compressed. Only if the compressed size is smaller than the serialized size will the sample be stored and sent compressed on the wire.

For batching we check the maximum serialized size of the batch, calculated as serialized_sample_max_size * **rti::core::policy::Batch::max_samples** (p. 883)

[range] [0, 2147483647] or **dds::core::LENGTH_UNLIMITED** (p. 438)

Setting the threshold to **dds::core::LENGTH_UNLIMITED** (p. 438) disables the compression.

[default] rti::core::CompressionSettings::compression_threshold_default() (8192)

Note

Only available for a **dds::pub::DataWriter** (p. 1138) and **dds::topic::Topic** (p. 2474).

Categories

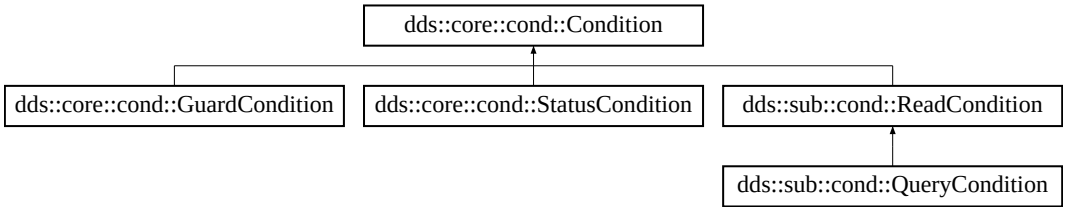
Immutable (p. 243), **Sending Large Data** (p. 197)

8.44 dds::core::cond::Condition Class Reference

<<***reference-type***>> *(p. 351)* Abstract base class of all the conditions

`#include <dds/core/cond/Condition.hpp>`

Inheritance diagram for dds::core::cond::Condition:



Public Member Functions

- void **dispatch** ()

  Dispatches the functors that have been registered with the condition.
- bool **trigger_value** () const

  *This operation retrieves the trigger_value of the **Condition** (p. 948).*

8.44.1 Detailed Description

<<***reference-type***>> *(p. 351)* Abstract base class of all the conditions

Note

Condition (p. 948) and its subclasses provide all the functions of a <<***reference-type***>> *(p. 351)* except **close()** (p. 1024) and retain().

This basic class is specialised in three classes:

dds::core::cond::GuardCondition (p. 1594), **dds::core::cond::StatusCondition** (p. 2372), and **dds::sub::cond::ReadCondition** (p. 2133).

A **dds::core::cond::Condition** (p. 948) has a `trigger_value` that can be true or false and is set automatically by RTI Connext.

See also

dds::core::cond::WaitSet (p. 2617)

Waitset Use Cases (p. 330)

8.44.2 Member Function Documentation

8.44.2.1 dispatch()

```
void dds::core::cond::Condition::dispatch ( )  [inline]
```

Dispatches the functors that have been registered with the condition.

If **trigger_value()** (p. 949) is true, calling **dispatch()** (p. 949) will call the registered functor handlers.

Note: it is more common to use **WaitSet::dispatch()** (p. 2624) rather than this function.

8.44.2.2 trigger_value()

```
bool dds::core::cond::Condition::trigger_value ( ) const  [inline]
```

This operation retrieves the trigger_value of the **Condition** (p. 948).

8.45 rti::queuing::ConsumerAvailabilityParams Class Reference

Definition of the availability feedback information that can be provided by consumers to Queuing Service.

```
#include <QueueSupport.hpp>
```

Public Attributes

- bool **reception_enabled**

  *Flag that provides a way to indicate whether or not the **QueueConsumer** (p. 2063) can receive samples from Queuing Service.*
- int **unacked_threshold**

  *Maximum number of samples pending acknowledgement that the **QueueConsumer** (p. 2063) can have to maintain sample reception. Note: This value is only applied when **ConsumerAvailabilityParams::reception_enabled** (p. 950) is true.*

8.45.1 Detailed Description

Definition of the availability feedback information that can be provided by consumers to Queuing Service.

8.45.2 Member Data Documentation

8.45.2.1 reception_enabled

```
bool rti::queuing::ConsumerAvailabilityParams::reception_enabled
```

Flag that provides a way to indicate whether or not the **QueueConsumer** (p. 2063) can receive samples from Queuing Service.

This value acts as a switch that QueueConsumers can use to temporarily enable or disable the reception of samples from Queuing Service.

8.45.2.2 unacked_threshold

```
int rti::queuing::ConsumerAvailabilityParams::unacked_threshold
```

Maximum number of samples pending acknowledgement that the **QueueConsumer** (p. 2063) can have to maintain sample reception. Note: This value is only applied when **ConsumerAvailabilityParams::reception_enabled** (p. 950) is true.

Once this value is reached, the **QueueConsumer** (p. 2063) will not receive more samples until it acknowledged enough samples so that it is one again under the unacked threshold.

This value has precedence over the unacked threshold settings defined in the Queuing Service configuration. This allows you to dynamically change the threshold on a per Consumer basis.

8.46 rti::topic::ContentFilter< T, CompileData > Class Template Reference

<<***extension***>> *(p. 354)* A class to inherit from when implementing a custom content filter

```
#include <rti/topic/ContentFilter.hpp>
```

Public Member Functions

- virtual CompileData & **compile** (const std::string &expression, const **dds::core::StringSeq** &parameters, const **dds::core::optional**< **dds::core::xtypes::DynamicType** > &type_code, const std::string &type_class_name, CompileData *old_compile_data)=0

  Compile an instance of the content filter according to the filter expression and parameters of the given data type.
- virtual bool **evaluate** (CompileData &compile_data, const T &sample, const **FilterSampleInfo** &meta_data)=0

  Evaluate whether the sample is passing the filter or not according to the sample content.
- virtual void **finalize** (CompileData &compile_data)=0

  A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

Related Functions

(Note that these are not member functions.)

- **no_compile_data_t no_compile_data**

  A constant to return if your compile function does not create any compile data

8.46.1 Detailed Description

template<typename T, typename CompileData = no_compile_data_t>
class rti::topic::ContentFilter< T, CompileData >

<<***extension***>> *(p. 354)* A class to inherit from when implementing a custom content filter

This interface can be implemented by an application-provided class and then registered with the DomainParticipant such that samples can be filtered for **dds::topic::ContentFilteredTopic** (p. 954) with the filter name that the filter is registered with.

Note: the API for using a custom content filter is subject to change in a future release.

For an example of how to create a custom content filter see **Creating Custom Content Filters** (p. 333)

Template Parameters

T	The type of the samples that this **ContentFilter** (p. 950) will filter
CompileData	the type of the data that the compile function will return. If your compile function will not return data, you can leave this template parameter to the default **no_compile_data_t** (p. 1835) type and return rti::topic::no_compile_data in your compile function

See also

rti::topic::CustomFilter (p. 967)

dds::domain::DomainParticipant::register_contentfilter() (p. 1349)

8.46.2 Member Function Documentation

8.46.2.1 compile()

```
template<typename T , typename CompileData = no_compile_data_t>
virtual CompileData &  rti::topic::ContentFilter< T, CompileData >::compile (
            const std::string & expression,
            const  dds::core::StringSeq & parameters,
            const  dds::core::optional<  dds::core::xtypes::DynamicType > & type_code,
            const std::string & type_class_name,
            CompileData * old_compile_data )  [pure virtual]
```

Compile an instance of the content filter according to the filter expression and parameters of the given data type.

This method is called when an instance of the locally registered content filter is created or when the expression parameter for the locally registered content filter instance is changed.

An instance of the locally registered content filter is created every time a local **dds::topic::ContentFilteredTopic** (p. 954) with the matching filter name is created, or when a **dds::sub::DataReader** (p. 976) with a matching filter name is discovered.

It is possible for multiple threads to be calling into this function at the same time. However, this function will never be called on a content filter that has been unregistered.

Parameters

expression	An std::string with the filter expression. If you want to manipulate this string, you **must** first make a copy of it.
parameters	A string sequence with the expression parameters the ContentFilteredTopic was created with. The string sequence is **equal** (but **not** identical) to the string sequence passed to **dds::topic::ContentFilteredTopic()** (p. 954). Note that the sequence passed to the compile function is **owned** by RTI Connext and **must not** be referenced outside the compile function.
type_code	The DynamicType for the related Topic of the ContentFilteredTopic. A type code is a description of a type in terms of which **types** it contains (int32, string, etc.) and the corresponding member field names in the data type structure. The type code can be used to write custom content filters that can be used with any type.
type_class_name	Fully qualified class name of the related Topic.
old_compile_data	The previous CompileData value from a previous call to this instance of a content filter. If the compile function is called more than once for an instance of a ContentFilteredTopic, e.g., if the expression parameters are changed, then the CompileData value returned by the previous invocation is passed in the old_compile_data parameter (which can be NULL). If this is a new instance of the filter, NULL is passed. This parameter is useful for freeing or reusing previously allocated resources.

Returns

User specified compile data for this instance of the content filter. This value is then passed to the evaluate finalize functions for this instance of the content filter. Can be set to rti::topic::no_compile_data.

8.46.2.2 evaluate()

```
template<typename T , typename CompileData = no_compile_data_t>
virtual bool  rti::topic::ContentFilter< T, CompileData >::evaluate (
             CompileData & compile_data,
             const T & sample,
             const  FilterSampleInfo & meta_data )  [pure virtual]
```

Evaluate whether the sample is passing the filter or not according to the sample content.

This method is called when a sample for a locally created **dds::sub::DataReader** (p. 976) associated with the filter is received, or when a sample for a discovered **dds::sub::DataReader** (p. 976) assocated with the filter needs to be sent.

It is possible for multiple threads to be calling into this function at the same time. However, this function will never be called on a content filter that has been unregistered.

Parameters

compile_data	The last return value of the compile function for this instance of the content filter.
sample	A deserialized sample to be filtered.
meta_data	Meta data associated with the sample.

Returns

The function must return false if the sample should be filtered out (did not pass the filter), true otherwise

8.46.2.3 finalize()

```
template<typename T , typename CompileData = no_compile_data_t>
virtual void  rti::topic::ContentFilter< T, CompileData >::finalize (
             CompileData & compile_data )  [pure virtual]
```

A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

This method is called when an instance of the locally registered content filter is deleted.

An instance of the locally registered content filter is deleted every time a local **dds::topic::ContentFilteredTopic** (p. 954) with the matching filter name is deleted, or when a **dds::sub::DataReader** (p. 976) with a matching filter name is removed due to discovery.

This method is also called on all instances of the discovered **dds::sub::DataReader** (p. 976) with a matching filter name if the filter is unregistered with **dds::domain::DomainParticipant::unregister_contentfilter(const std::string & filter_name)** (p. 1350).

It is possible for multiple threads to be calling into this function at the same time. However, this function will never be called on a content filter that has been unregistered.

Parameters

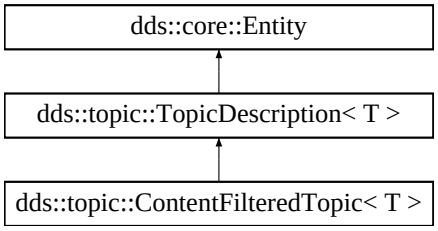
compile_data	The last return value of the compile function for this instance of the content filter.

8.47 dds::topic::ContentFilteredTopic< T > Class Template Reference

<<***reference-type***>> *(p. 351)* Specialization of **TopicDescription** (p. 2503) that allows for content-based subscriptions.

`#include <dds/topic/ContentFilteredTopic.hpp>`

Inheritance diagram for dds::topic::ContentFilteredTopic< T >:



Public Member Functions

- **ContentFilteredTopic** (const **Topic**< T > &the_topic, RTI_STRING_IN the_name, const **dds::topic::Filter** & **filter**)

  *Creates a **ContentFilteredTopic** (p. 954) to perform content-based subscriptions.*
- std::string **filter_expression** () const

  Gets the filter expression.
- const **dds::core::StringSeq** **filter_parameters** () const

  Gets the filter expression parameters.
- template<typename FwdIterator >
  void **filter_parameters** (const FwdIterator &begin, const FwdIterator &end)

  Modifies the filter parameters.
- const **dds::topic::Topic**< T > & **topic** () const

  Gets the related topic.
- void **filter** (const **dds::topic::Filter** &the_filter)

  <<***extension***>> *(p. 354) Modifies the filter*
- void **append_to_expression_parameter** (int32_t index, const std::string &val)

  <<***extension***>> *(p. 354) Appends a term to a parameter*
- void **remove_from_expression_parameter** (int32_t index, const std::string &val)

  <<***extension***>> *(p. 354) Removes a term from a parameter*

Related Functions

(Note that these are not member functions.)

- template<typename BinIterator >
  uint32_t **find_registered_content_filters** (const **dds::domain::DomainParticipant** & **participant**, BinIterator begin)

  *Lookup the names of all of the custom content filters registered to a **dds::domain::DomainParticipant*** (p. *1326*).

8.47.1 Detailed Description

template<typename T>
class dds::topic::ContentFilteredTopic< T >

<<***reference-type***>> *(p. 351)* Specialization of **TopicDescription** (p. 2503) that allows for content-based subscriptions.

It describes a more sophisticated subscription that indicates a **dds::sub::DataReader** (p. 976) does not want to necessarily see all values of each instance published under the **dds::topic::Topic** (p. 2474). Rather, it wants to see only the values whose contents satisfy certain criteria. This class therefore can be used to request content-based subscriptions.

The selection of the content is done using the `filter_expression` with parameters `expression_parameters`.

- The `filter_expression` attribute is a string that specifies the criteria to select the data samples of interest. It is similar to the WHERE part of an SQL clause.

- The `expression_parameters` attribute is a sequence of strings that give values to the 'parameters' (i.e. "%n" tokens) in the `filter_expression`. The number of supplied parameters must fit with the requested values in the `filter_expression` (i.e. the number of n tokens).

Queries and Filters Syntax (p. 84) describes the syntax of `filter_expression` and `expression_parameters`.

Template Parameters

T	The topic-type

See also

Filtering with ContentFilteredTopic (p. 332)
Creating Custom Content Filters (p. 333)

8.47.2 Constructor & Destructor Documentation

8.47.2.1 ContentFilteredTopic()

```
template<typename T >
dds::topic::ContentFilteredTopic< T >::ContentFilteredTopic (
            const  Topic< T > & the_topic,
            RTI_STRING_IN the_name,
            const  dds::topic::Filter & filter )  [inline]
```

Creates a **ContentFilteredTopic** (p. 954) to perform content-based subscriptions.

The **ContentFilteredTopic** (p. 954) only relates to samples published under that **Topic** (p. 2474), filtered according to their content. The filtering is done by means of evaluating a logical expression that involves the values of some of the data-fields in the sample. The logical expression derived from the **Filter** (p. 1558) arguments.

Queries and Filters Syntax (p. 84) describes the syntax of filter_expression and expression_parameters.

Precondition

The application is not allowed to create two **ContentFilteredTopic** (p. 954) objects with the same topic name attached to the same DomainParticipant. If the application attempts this, this function will fail and throw **dds::core::Error** (p. 1539).

By default this function will create a content filter using the built-in SQL filter which implements a superset of the DDS specification. This filter requires that all IDL types have been compiled with DynamicType (also known as TypeCode). If this precondition is not met, this operation return throws **dds::core::Error** (p. 1539). Do not use rtiddsgen's -notypecode option if you want to use the built-in SQL filter.

To use a different filter, set **Filter::name()** (p. 1562).

Parameters

the_topic	The **Topic** (p. 2474) to be filtered
the_name	Name for the new content filtered topic, must not exceed 255 characters.
filter	The filter to apply, which includes the filter name (by default SQL), the filter expression and the expression parameters.

8.47.3 Member Function Documentation

8.47.3.1 filter_expression()

```
template<typename T >
std::string  dds::topic::ContentFilteredTopic< T >::filter_expression ( ) const  [inline]
```

Gets the filter expression.

8.47.3.2 filter_parameters() [1/2]

```
template<typename T >
const  dds::core::StringSeq  dds::topic::ContentFilteredTopic< T >::filter_parameters ( ) const
[inline]
```

Gets the filter expression parameters.

8.47.3.3 filter_parameters() [2/2]

```
template<typename T >
template<typename FwdIterator >
void  dds::topic::ContentFilteredTopic< T >::filter_parameters (
             const FwdIterator & begin,
             const FwdIterator & end )  [inline]
```

Modifies the filter parameters.

Template Parameters

FwdIterator	A forward iterator whose value type is std::string (or convertible to std::string)

Parameters

begin	The beginning of the range
end	The end of the range (can't contain more than 100 elements)

8.47.3.4 topic()

```
template<typename T >
const  dds::topic::Topic< T > &  dds::topic::ContentFilteredTopic< T >::topic ( ) const  [inline]
```

Gets the related topic.

8.47.3.5 filter()

```
template<typename T >
void filter (
             const  dds::topic::Filter & the_filter )
```

<<***extension***>> *(*p. *354)* Modifies the filter

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Parameters

the_filter	Contains the new filter expression and parameters. The filter name is immutable: if filter->**name()** (p. 2504) is different it will be ignored.

Changes the `filter_expression` and `expression_parameters` associated with the **dds::topic::Content↩ FilteredTopic** (p. 954).

8.47.3.6 append_to_expression_parameter()

```
template<typename T >
void append_to_expression_parameter (
             int32_t index,
             const std::string & val )
```

<<***extension***>> *(p. 354)* Appends a term to a parameter

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Appends the input string to the end of the specified parameter string, separated by a comma. If the original parameter string is enclosed in quotation marks ("), the resultant string will also be enclosed in quotation marks.

This method can be used in expression parameters associated with MATCH operators in order to add a pattern to the match pattern list. For example, if the filter expression parameter value is:

'IBM'

Then append_to_expression_parameter(0, "MSFT") would generate the new value:

'IBM,MSFT'

Parameters

index	<<***in***>> *(p. 355)* The index of the parameter string to be modified. The first index is index 0. When using the **rti::topic::stringmatch_filter_name** (p. 49) filter, `index` *must* be 0.
val	<<***in***>> *(p. 355)* The string term to be appended to the parameter string.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.47.3.7 remove_from_expression_parameter()

```
template<typename T >
void remove_from_expression_parameter (
            int32_t index,
            const std::string & val )
```

<<***extension***>> *(p. 354)* Removes a term from a parameter

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Removes the input string from the specified parameter string. To be found and removed, the input string must exist as a complete term, bounded by comma separators or the strong boundary. If the original parameter string is enclosed in quotation marks ("), the resultant string will also be enclosed in quotation marks. If the removed term was the last entry in the string, the result will be a string of empty quotation marks.

This method can be used in expression parameters associated with MATCH operators in order to remove a pattern from the match pattern list. For example, if the filter expression paremeter value is:

'IBM,MSFT'

Then remove_from_expression_parameter(0, "IBM") would generate the expression:

'MSFT'

Parameters

index	<<***in***>> *(p. 355)* The index of the parameter string to be modified. The first index is index 0. When using the **rti::topic::stringmatch_filter_name** (p. 49) filter, `index` *must* be 0.
val	<<***in***>> *(p. 355)* The string term to be removed from the parameter string.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.47.4 Friends And Related Function Documentation

8.47.4.1 find_registered_content_filters()

```
template<typename BinIterator >
uint32_t find_registered_content_filters (
```

```
const  dds::domain::DomainParticipant & participant,
BinIterator begin )  [related]
```

Lookup the names of all of the custom content filters registered to a **dds::domain::DomainParticipant** (p. 1326).

```
#include <dds/topic/find.hpp>
```

Note

This is a standalone function in the namespace **rti::topic** (p. 771)

The names of the RTI Connext built-in content filters will not be returned as part of the list.

Template Parameters

BinIterator	A back-inserting iterator whose value type is `std::string` (or convertible to)

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) that the content filters are registered with
begin	A back-inserting iterator to the position in the destination container to insert the the names of the found content filters into

Returns

The number of found content filters

References **rti::topic::find_registered_content_filters()**.

8.48 rti::core::ContentFilterProperty Class Reference

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Provides all the required information to enable content filtering.

```
#include <ContentFilterProperty.hpp>
```

Public Member Functions

- **optional_string content_filter_topic_name** () const
  Name of the ContentFilteredTopic associated with the DataReader.
- **dds::core::StringSeq expression_parameters** () const
  The ContentFilteredTopic filter parameters.
- **optional_string filter_class_name** () const
  Identifies the filter class this filter belongs to.
- **optional_string filter_expression** () const
  The filter expression.
- **optional_string related_topic_name** () const
  The name of the ContentFilteredTopic's related Topic.

8.48.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Provides all the required information to enable content filtering.

8.48.2 Member Function Documentation

8.48.2.1 content_filter_topic_name()

```
optional_string rti::core::ContentFilterProperty::content_filter_topic_name ( ) const  [inline]
```

Name of the ContentFilteredTopic associated with the DataReader.

8.48.2.2 expression_parameters()

```
dds::core::StringSeq rti::core::ContentFilterProperty::expression_parameters ( ) const  [inline]
```

The ContentFilteredTopic filter parameters.

8.48.2.3 filter_class_name()

```
optional_string rti::core::ContentFilterProperty::filter_class_name ( ) const  [inline]
```

Identifies the filter class this filter belongs to.

8.48.2.4 filter_expression()

```
optional_string rti::core::ContentFilterProperty::filter_expression ( ) const  [inline]
```

The filter expression.

8.48.2.5 related_topic_name()

```
optional_string rti::core::ContentFilterProperty::related_topic_name ( ) const  [inline]
```

The name of the ContentFilteredTopic's related Topic.

8.49 rti::util::network_capture::ContentKindMask Class Reference

<<***extension***>> *(p. 354)* Mask indicating the types of contents to remove from RTPS frames before saving them to the capture file.

```
#include <network_capture.hpp>
```

Inherits std::bitset< 32 >.

Public Types

- typedef std::bitset< 32 > **MaskType**

  A typedef of std::bitset<32> for convenience.

Public Member Functions

- **ContentKindMask** ()

  *Default constructor for **ContentKindMask** (p. 962).*
- **ContentKindMask** (uint64_t mask)

  *Construct a **ContentKindMask** (p. 962) from an integer.*
- **ContentKindMask** (const **MaskType** &mask)

  *Construct a **ContentKindMask** (p. 962) from a MaskType object.*

Static Public Member Functions

- static const **ContentKindMask** **user** ()

  The serialized data coming from a user.
- static const **ContentKindMask** **encrypted** ()

  The encrypted user data.
- static const **ContentKindMask** **default_mask** ()

  *Default mask for **network_capture::ContentKindMask** (p. 962): do not remove any content.*
- static const **ContentKindMask** **none** ()

  The RTPS frames in the capture file will be saved as they are.
- static const **ContentKindMask** **all** ()

  The RTPS frames in the capture file will not include user data (either plain or encrypted).

8.49.1 Detailed Description

<<***extension***>> *(p. 354)* Mask indicating the types of contents to remove from RTPS frames before saving them to the capture file.

The masks are based on a combination (or only one) of the **network_capture::ContentKindMask** (p. 962) bitmaps.

See also

network_capture::ContentKindMask (p. 962)

8.49.2 Member Typedef Documentation

8.49.2.1 MaskType

```
typedef std::bitset<32>  rti::util::network_capture::ContentKindMask::MaskType
```

A typedef of std::bitset<32> for convenience.

8.49.3 Constructor & Destructor Documentation

8.49.3.1 ContentKindMask() [1/3]

```
rti::util::network_capture::ContentKindMask::ContentKindMask ( )  [inline]
```

Default constructor for **ContentKindMask** (p. 962).

8.49.3.2 ContentKindMask() [2/3]

```
rti::util::network_capture::ContentKindMask::ContentKindMask (
            uint64_t mask )  [inline], [explicit]
```

Construct a **ContentKindMask** (p. 962) from an integer.

Parameters

mask	Value whose bits are copied to the bitset positions

8.49.3.3 ContentKindMask() [3/3]

```
rti::util::network_capture::ContentKindMask::ContentKindMask (
            const  MaskType & mask )  [inline]
```

Construct a **ContentKindMask** (p. 962) from a MaskType object.

Parameters

mask	A std::bitset<32> to construct this NetworkCaptureContentKindMask from.

8.49.4 Member Function Documentation

8.49.4.1 user()

```
static const ContentKindMask rti::util::network_capture::ContentKindMask::user ( ) [inline],
[static]
```

The serialized data coming from a user.

8.49.4.2 encrypted()

```
static const ContentKindMask rti::util::network_capture::ContentKindMask::encrypted ( ) [inline],
[static]
```

The encrypted user data.

8.49.4.3 default_mask()

```
static const ContentKindMask rti::util::network_capture::ContentKindMask::default_mask ( ) [inline],
[static]
```

Default mask for **network_capture::ContentKindMask** (p. 962): do not remove any content.

It is equivalent to **network_capture::ContentKindMask::none()** (p. 964).

[default] Do not remove any content.

8.49.4.4 none()

```
static const ContentKindMask rti::util::network_capture::ContentKindMask::none ( ) [inline],
[static]
```

The RTPS frames in the capture file will be saved as they are.

8.49.4.5 all()

```
static const ContentKindMask rti::util::network_capture::ContentKindMask::all ( ) [inline],
[static]
```

The RTPS frames in the capture file will not include user data (either plain or encrypted).

Its value is the result of setting the bits for removing user data and removing encrypted data: (**network_capture::ContentKindMask::user()** (p. 964)) | **network_capture::ContentKindMask::encrypted()** (p. 964))

8.50 rti::core::Cookie Class Reference

Unique identifier for a written data sample in the form of a sequence of bytes.

```
#include <rti/core/Cookie.hpp>
```

Public Member Functions

- **Cookie** ()

  Creates an empty cookie.
- template<typename ByteContainer >
  Cookie (const ByteContainer &bytes)

  Creates a new cookie with the bytes inside a container.
- **dds::core::vector**< uint8_t > & **value** ()

  Retrieves a reference to the vector of bytes.
- const **dds::core::vector**< uint8_t > & **value** () const

  Retrieves a const reference to the vector of bytes.
- template<typename T >
  T * **to_pointer** () const

  Converts a cookie into a pointer.

Related Functions

(Note that these are not member functions.)

- typedef **dds::core::vector**< **Cookie** > **CookieSeq**

  *A sequence of **Cookie** (p. 965) objects.*
- std::ostream & **operator**<< (std::ostream &out, const **Cookie** &cookie)

  *Prints a **Cookie** (p. 965) to an output stream.*

8.50.1 Detailed Description

Unique identifier for a written data sample in the form of a sequence of bytes.

See also

dds::pub::DataWriter::write(const T&, rti::pub::WriteParams&) (p. 1179)

8.50.2 Constructor & Destructor Documentation

8.50.2.1 Cookie() [1/2]

```
rti::core::Cookie::Cookie ( )  [inline]
```

Creates an empty cookie.

8.50.2.2 Cookie() [2/2]

```
template<typename ByteContainer >
rti::core::Cookie::Cookie (
            const ByteContainer & bytes )  [inline], [explicit]
```

Creates a new cookie with the bytes inside a container.

8.50.3 Member Function Documentation

8.50.3.1 value() [1/2]

```
dds::core::vector< uint8_t > & rti::core::Cookie::value ( )
```

Retrieves a reference to the vector of bytes.

8.50.3.2 value() [2/2]

```
const  dds::core::vector< uint8_t > & rti::core::Cookie::value ( ) const
```

Retrieves a const reference to the vector of bytes.

8.50.3.3 to_pointer()

```
template<typename T >
T * rti::core::Cookie::to_pointer ( ) const  [inline]
```

Converts a cookie into a pointer.

Utility function that allows retrieving a pointer stored in a cookie.

Precondition

The cookie's value was filled with a pointer, otherwise the value returned may not be a valid memory address.

Template Parameters

T	The pointer type

Returns

The pointer this cookie represents

8.50.4 Friends And Related Function Documentation

8.50.4.1 CookieSeq

```
typedef  dds::core::vector< Cookie>  CookieSeq  [related]
```

A sequence of **Cookie** (p. 965) objects.

8.50.4.2 operator<<()

```
std::ostream & operator<< (
             std::ostream & out,
             const  Cookie & cookie )  [related]
```

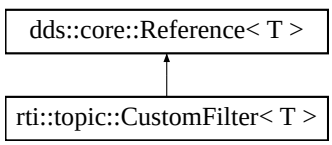
Prints a **Cookie** (p. 965) to an output stream.

8.51 rti::topic::CustomFilter< T > Class Template Reference

<<***extension***>> *(p. 354)* <<***reference-type***>> *(p. 351)* A wrapper class for the user-defined implementation of a **ContentFilter** (p. 950).

```
#include <rti/topic/CustomFilter.hpp>
```

Inheritance diagram for rti::topic::CustomFilter< T >:



Public Member Functions

- **CustomFilter** (typename dds::core::smart_ptr_traits< T >::ref_type content_filter_ref)

  *Create a **CustomFilter** (p. 967) which holds a shared pointer to a user-defined content filter.*
- const T * **get** () const

  Get a const pointer to the underlying content filter.
- T * **get** ()

  Get a pointer to the underlying content filter.

8.51.1 Detailed Description

template<typename T>
class rti::topic::CustomFilter< T >

*<<**extension**>> (p. 354) <<**reference-type**>> (p. 351)* A wrapper class for the user-defined implementation of a **ContentFilter** (p. 950).

In order to register a custom content filter a user must:

- Implement the content filter by creating their own class which inherits from one of the abstract classes: **ContentFilter** (p. 950), **WriterContentFilter** (p. 2652), or **WriterContentFilterHelper** (p. 2657)
- Create a **CustomFilter** (p. 967) which holds a shared pointer to the user-defined content filter
- Register the **CustomFilter** (p. 967) with the **dds::domain::DomainParticipant** (p. 1326) using **dds::domain::DomainParticipant::register_contentfilter** (p. 1349)

The **CustomFilter** (p. 967) class prevents your content filter from being deleted while RTI Connext is using your filter by retaining a reference to your filter until it is no longer being used by RTI Connext.

For an example of how to create anad register a custom content filter see **Creating Custom Content Filters** (p. 333).

Template Parameters

T	The type of the custom content filter that is being registered. It must be a type which inherits and implements the interface of one of **ContentFilter** (p. 950), **WriterContentFilter** (p. 2652), or **WriterContentFilterHelper** (p. 2657)

8.51.2 Constructor & Destructor Documentation

8.51.2.1 CustomFilter()

```
template<typename T >
rti::topic::CustomFilter< T >::CustomFilter (
            typename dds::core::smart_ptr_traits< T >::ref_type content_filter_ref )  [inline]
```

Create a **CustomFilter** (p. 967) which holds a shared pointer to a user-defined content filter.

Parameters

content_filter_ref	A shared pointer to a user-defined content filter object.

8.51.3 Member Function Documentation

8.51.3.1 get() [1/2]

```
template<typename T >
const T *  rti::topic::CustomFilter< T >::get ( ) const  [inline]
```

Get a const pointer to the underlying content filter.

8.51.3.2 get() [2/2]

```
template<typename T >
T *  rti::topic::CustomFilter< T >::get ( )  [inline]
```

Get a pointer to the underlying content filter.

8.52 rti::core::policy::Database Class Reference

<<***extension***>> *(p. 116)* Configures threads and resource limits that RTI Connext uses to control its internal database

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **Database** & **thread** (const **rti::core::ThreadSettings** &the_thread)

  Sets the thread settings.
- const **rti::core::ThreadSettings** & **thread** () const

  Gets the thread settings (by const reference)
- **rti::core::ThreadSettings** & **thread** ()

  Gets the thread settings by reference.
- **Database** & **shutdown_timeout** (const **dds::core::Duration** &the_shutdown_timeout)

  [DEPRECATED] *(*p. ***296****) The maximum wait time during a shutdown.*
- **dds::core::Duration** **shutdown_timeout** () const

  Getter (see setter with the same name)
- **Database** & **cleanup_period** (const **dds::core::Duration** &the_cleanup_period)

  [DEPRECATED] *(*p. ***296****) The period at which the database thread wakes up to remove deleted records.*
- **dds::core::Duration** **cleanup_period** () const

  Getter (see setter with the same name)
- **Database** & **shutdown_cleanup_period** (const **dds::core::Duration** &period)

  [DEPRECATED] *(*p. ***296****) The clean-up period used during database shut-down.*
- **dds::core::Duration** **shutdown_cleanup_period** () const

  Getter (see setter with the same name)
- **Database** & **initial_records** (int32_t the_initial_records)
- int32_t **initial_records** () const

  Getter (see setter with the same name)
- **Database** & **max_skiplist_level** (int32_t the_max_skiplist_level)

  [DEPRECATED] *(*p. ***296****) The maximum level of the skiplist.*
- **Database** & **max_weak_references** (int32_t the_max_weak_references)
- int32_t **max_weak_references** () const

  Getter (see setter with the same name)
- **Database** & **initial_weak_references** (int32_t the_initial_weak_references)
- int32_t **initial_weak_references** () const

  Getter (see setter with the same name)

8.52.1 Detailed Description

<<***extension***>> *(*p. *116**)* Configures threads and resource limits that RTI Connext uses to control its internal database

RTI uses an internal in-memory "database" to store information about entities created locally as well as remote entities found during the discovery process. This database uses a background thread to garbage-collect records related to deleted entities. When the **dds::domain::DomainParticipant** (p. 1326) that maintains this database is deleted, it shuts down this thread.

The **Database** (p. 969) QoS policy is used to configure how RTI Connext manages its database, including how often it cleans up, the priority of the database thread, and limits on resources that may be allocated by the database.

You may be interested in modifying the **rti::core::policy::Database::shutdown_timeout** (p. 972) and **rti::core::policy::Database::shutdown_cleanup_period** (p. 973) parameters to decrease the time it takes to delete a **dds::domain::DomainParticipant** (p. 1326) when your application is shutting down.

The **rti::core::policy::DomainParticipantResourceLimits** (p. 1391) controls the memory allocation for elements stored in the database.

This QoS policy is an extension to the DDS standard.

Entity:

dds::domain::DomainParticipant (p. 1326)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) **NO** (p. 514)

8.52.2 Member Function Documentation

8.52.2.1 thread() [1/3]

```
Database & rti::core::policy::Database::thread (
            const  rti::core::ThreadSettings & the_thread )
```

Sets the thread settings.

There is only one database thread: the clean-up thread.

[default] Priority: LOW.
The actual value depends on your architecture:

For Windows: -3
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] Stack Size: The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] Mask: **rti::core::ThreadSettingsKindMask::stdio()** (p. 2456)

Categories

Immutable (p. 243), **Concurrency** (p. 164)

8.52.2.2 thread() [2/3]

```
const  rti::core::ThreadSettings & rti::core::policy::Database::thread ( ) const
```

Gets the thread settings (by const reference)

8.52.2.3 thread() [3/3]

```
rti::core::ThreadSettings & rti::core::policy::Database::thread ( )
```

Gets the thread settings by reference.

See also

thread(const rti::core::ThreadSettings&) (p. 971)

8.52.2.4 shutdown_timeout() [1/2]

```
Database & rti::core::policy::Database::shutdown_timeout (
            const  dds::core::Duration & the_shutdown_timeout )
```

[DEPRECATED] (p. **296)** The maximum wait time during a shutdown.

The domain participant will exit after the timeout, even if the database has not been fully cleaned up.

[default] 15 seconds

[range] [0,**dds::core::Duration::infinite()** (p. 1451)]

Categories

Immutable (p. 243)

8.52.2.5 shutdown_timeout() [2/2]

```
dds::core::Duration rti::core::policy::Database::shutdown_timeout ( ) const
```

Getter (see setter with the same name)

8.52.2.6 cleanup_period() [1/2]

```
Database & rti::core::policy::Database::cleanup_period (
            const  dds::core::Duration & the_cleanup_period )
```

[DEPRECATED] (p. **296)** The period at which the database thread wakes up to remove deleted records.

[default] 61 seconds

[range] [0,1 year]

Categories

Immutable (p. 243)

8.52.2.7 cleanup_period() [2/2]

```
dds::core::Duration rti::core::policy::Database::cleanup_period ( ) const
```

Getter (see setter with the same name)

8.52.2.8 shutdown_cleanup_period() [1/2]

```
Database & rti::core::policy::Database::shutdown_cleanup_period (
            const  dds::core::Duration & period )
```

[DEPRECATED] (p. **296)** The clean-up period used during database shut-down.

If you would like to shorten the time taken for a DomainParticipant to shutdown, you can decrease this value.

It is recommended to set this value to something other than 0 if running in an RTOS environment, to avoid CPU starvation.

[default] 10 milliseconds

[range] [0,1 year]

Categories

Immutable (p. 243)

8.52.2.9 shutdown_cleanup_period() [2/2]

```
dds::core::Duration rti::core::policy::Database::shutdown_cleanup_period ( ) const
```

Getter (see setter with the same name)

8.52.2.10 initial_records() [1/2]

```
Database & rti::core::policy::Database::initial_records (
            int32_t the_initial_records )
```

[default] 1024

[range] [1,10 million]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.52.2.11 initial_records() [2/2]

```
int32_t rti::core::policy::Database::initial_records ( ) const
```

Getter (see setter with the same name)

8.52.2.12 max_skiplist_level()

```
Database & rti::core::policy::Database::max_skiplist_level (
            int32_t the_max_skiplist_level )
```

[DEPRECATED] (p. 296) The maximum level of the skiplist.

The skiplist is used to keep records in the database. Usually, the search time is log2(N), where N is the total number of records in one skiplist. However, once N exceeds 2^n, where n is the maximum skiplist level, the search time will become more and more linear. Therefore, the maximum level should be set such that 2^n is larger than the maximum(N among all skiplists). Usually, the maximum N is the maximum number of remote and local writers or readers.

[default] 7

[range] [1,31]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.52.2.13 max_weak_references() [1/2]

```
Database & rti::core::policy::Database::max_weak_references (
            int32_t the_max_weak_references )
```

A weak reference is an internal data structure that refers to a record within RTI Connext' internal database. This field configures the maximum number of such references that RTI Connext may create.

The actual number of weak references is permitted to grow from an initial value (indicated by **rti::core::policy::↩ Database::initial_weak_references** (p. 975)) to this maximum. To prevent RTI Connext from allocating any weak references after the system has reached a steady state, set the initial and maximum values equal to one another. To indicate that the number of weak references should continue to grow as needed indefinitely, set this field to **dds::core↩ ::LENGTH_UNLIMITED** (p. 438). Be aware that although a single weak reference occupies very little memory, allocating a very large number of them can have a significant impact on your overall memory usage.

Tuning this value precisely is difficult without intimate knowledge of the structure of RTI Connext' database; doing so is an advanced feature not required by most applications. The default value has been chosen to be sufficient for reasonably large systems. If you believe you may need to modify this value, please consult with RTI support personnel for assistance.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 100 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_weak_references

See also

rti::core::policy::Database::initial_weak_references (p. 975)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.52.2.14 max_weak_references() [2/2]

```
int32_t rti::core::policy::Database::max_weak_references ( ) const
```

Getter (see setter with the same name)

8.52.2.15 initial_weak_references() [1/2]

```
Database & rti::core::policy::Database::initial_weak_references (
            int32_t the_initial_weak_references )
```

See **rti::core::policy::Database::max_weak_references** (p. 974) for more information about what a weak reference is.

If the QoS set contains an initial_weak_references value that is too small to ever grow to **rti::core::policy::Database↩ ::max_weak_references** (p. 974) using RTI Connext' internal algorithm, this value will be adjusted upwards as necessary. Subsequent accesses of this value will reveal the actual initial value used.

Changing the value of this field is an advanced feature; it is recommended that you consult with RTI support personnel before doing so.

[default] 2049, which is the minimum initial value imposed by RTI Connext when the maximum is unlimited. If a lower value is specified, it will simply be increased to 2049 automatically.

[range] [1, 100 million], <= max_weak_references

See also

rti::core::policy::Database::max_weak_references (p. 974)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.52.2.16 initial_weak_references() [2/2]

```
int32_t rti::core::policy::Database::initial_weak_references ( ) const
```

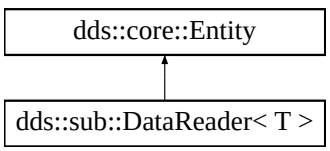
Getter (see setter with the same name)

8.53 dds::sub::DataReader< T > Class Template Reference

<<***reference-type***>> *(p. 351)* Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **dds::sub::Subscriber** (p. 2410).

```
#include "dds/sub/DataReader.hpp"
```

Inheritance diagram for dds::sub::DataReader< T >:



Classes

- class **ManipulatorSelector**

  *A **Selector** (p. 2320) class enabling the streaming API.*
- class **Selector**

  *The **Selector** (p. 2320) class is used by the **DataReader** (p. 976) to compose read and take operations.*

Public Member Functions

- **DataReader** (const **dds::sub::Subscriber** &sub, const **dds::topic::Topic**< T > &topic)

  *Create a **DataReader** (p. 976).*
- **DataReader** (const **dds::sub::Subscriber** &sub, const **dds::topic::Topic**< T > &topic, const **dds::sub::qos::DataReaderQos** &the_qos, **dds::sub::DataReaderListener**< T > *the_listener=NULL, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Create a **DataReader** (p. 976).*
- **DataReader** (const **dds::sub::Subscriber** &sub, const **dds::topic::Topic**< T > &topic, const **dds::sub::qos::DataReaderQos** &the_qos, std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Create a **DataReader** (p. 976).*
- **DataReader** (const **dds::sub::Subscriber** &sub, const **dds::topic::ContentFilteredTopic**< T > &topic)

  *Create a **DataReader** (p. 976) for a **dds::topic::ContentFilteredTopic** (p. 954).*
- **DataReader** (const **dds::sub::Subscriber** &sub, const **dds::topic::ContentFilteredTopic**< T > &topic, const **dds::sub::qos::DataReaderQos** &the_qos, **dds::sub::DataReaderListener**< T > *the_listener=NULL, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Create a **DataReader** (p. 976) for a **dds::topic::ContentFilteredTopic** (p. 954).*
- **DataReader** (const **dds::sub::Subscriber** &sub, const **dds::topic::ContentFilteredTopic**< T > &topic, const **dds::sub::qos::DataReaderQos** &the_qos, std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Create a **DataReader** (p. 976) for a **dds::topic::ContentFilteredTopic** (p. 954) with a listener.*
- **DataReader** (const **dds::topic::Topic**< T > &topic)

  *Create a **DataReader** (p. 976).*
- **DataReader** (const **dds::topic::Topic**< T > &topic, const **dds::sub::qos::DataReaderQos** &the_qos)

  *Create a **DataReader** (p. 976) with QoS.*
- **DataReader** (const **dds::topic::ContentFilteredTopic**< T > &topic, const **dds::sub::qos::DataReaderQos** &the_qos)

  *Create a **DataReader** (p. 976) for a **dds::topic::ContentFilteredTopic** (p. 954).*
- **DataReader** (const **dds::topic::Topic**< T > &topic, const **dds::sub::qos::DataReaderQos** &the_qos, std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Create a **DataReader** (p. 976).*
- **DataReader** (const **dds::topic::ContentFilteredTopic**< T > &topic)

  *Create a **DataReader** (p. 976) for a **dds::topic::ContentFilteredTopic** (p. 954).*
- **DataReader** (const **dds::topic::ContentFilteredTopic**< T > &topic, const **dds::sub::qos::DataReaderQos** &the_qos, std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Create a **DataReader** (p. 976) for a **dds::topic::ContentFilteredTopic** (p. 954) with a listener.*
- const **dds::sub::status::DataState** & **default_filter_state** ()

  Returns the default state for read/take operations.

- **DataReader** & **default_filter_state** (const **dds::sub::status::DataState** & **state**)

  Set the default state filter for read/take operations.
- **DataReader** & **operator**>> (**dds::sub::LoanedSamples**< T > &ls)

  *Use the stream operator to take samples, placing them into a **LoanedSamples** (p. 1663) container.*
- **ManipulatorSelector** **operator**>> (bool(∗manipulator)(**ReadModeDummyType**))

  Use the stream operator to read or take samples.
- template<typename Functor >
  ManipulatorSelector **operator**>> (Functor f)

  Use the stream operator to dictate the behavior of the read or take operation.
- **LoanedSamples**< T > **read** ()

  Read all samples using the default filter state.
- **LoanedSamples**< T > **take** ()

  Take all samples using the default filter state.
- template<typename SamplesFWIterator >
  uint32_t **read** (SamplesFWIterator sfit, int32_t **max_samples**)

  Read up to max_samples samples using the default filter state.
- template<typename SamplesFWIterator >
  uint32_t **take** (SamplesFWIterator sfit, int32_t **max_samples**)

  Take up to max_samples samples using the default filter state.
- template<typename SamplesBIIterator >
  uint32_t **read** (SamplesBIIterator sbit)

  Read all samples available in the reader cache using the default filter state.
- template<typename SamplesBIIterator >
  uint32_t **take** (SamplesBIIterator sbit)

  Take all samples available in the reader cache samples using the default filter state.
- **Selector** **select** ()

  *Get a **Selector** (p. 2320) to perform complex data selections, such as per-instance selection, content and status filtering.*
- T & **key_value** (T &key_holder, const **dds::core::InstanceHandle** &handle)

  Retrieve the instance key that corresponds to an instance handle.
- **dds::topic::TopicInstance**< T > & **key_value** (**dds::topic::TopicInstance**< T > &key_holder, const **dds**↩**::core::InstanceHandle** &handle)

  Retrieve the instance key that corresponds to an instance handle.
- **dds::core::InstanceHandle** **lookup_instance** (const T &key_holder) const

  Retrieve the InstanceHandle that corresponds to an instance key holder.
- **dds::topic::TopicDescription**< DataType > **topic_description** () const

  *Returns the **dds::topic::TopicDescription** (p. 2503) associated with the **DataReader** (p. 976).*
- const **dds::sub::Subscriber** & **subscriber** () const

  *Returns the **Subscriber** (p. 2410) to which the **DataReader** (p. 976) belongs.*
- void **listener** (**Listener** ∗the_listener, const **dds::core::status::StatusMask** &event_mask)

  Sets the listener associated with this reader.
- **Listener** ∗ **listener** () const

  *Returns the listener currently associated with this **DataReader** (p. 976).*
- void **set_listener** (std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &event_↩mask)

  Sets the listener associated with this reader.
- void **set_listener** (std::shared_ptr< **Listener** > the_listener)

  Sets the listener associated with this reader.
- std::shared_ptr< **Listener** > **get_listener** () const

*Returns the listener currently associated with this **DataReader** (p. 976).*

- const **dds::sub::qos::DataReaderQos** **qos** () const

  Get the QoS associated with this reader.
- void **qos** (const **dds::sub::qos::DataReaderQos** &the_qos)

  Set the QoS associated with this reader.
- **DataReader** & **operator<<** (const **dds::sub::qos::DataReaderQos** &the_qos)

  Set the QoS associated with this reader.
- **DataReader** & **operator>>** (**dds::sub::qos::DataReaderQos** &the_qos)

  Get the QoS associated with this reader.
- void **wait_for_historical_data** (const **dds::core::Duration** &max_wait)

  Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind.
- **dds::core::status::LivelinessChangedStatus** **liveliness_changed_status** () const

  *Get the LivelinessChangedStatus for this **DataReader** (p. 976).*
- **dds::core::status::SampleRejectedStatus** **sample_rejected_status** () const

  *Get the SampleRejectedStatus for this **DataReader** (p. 976).*
- **dds::core::status::SampleLostStatus** **sample_lost_status** () const

  *Get the SampleLostStatus for this **DataReader** (p. 976).*
- **dds::core::status::RequestedDeadlineMissedStatus** **requested_deadline_missed_status** ()

  *Get the RequestedDeadlineMissedStatus for this **DataReader** (p. 976).*
- **dds::core::status::RequestedIncompatibleQosStatus** **requested_incompatible_qos_status** () const

  *Get the RequestedIncompatibleQosStatus for this **DataReader** (p. 976).*
- **dds::core::status::SubscriptionMatchedStatus** **subscription_matched_status** () const

  *Get the SubscriptionMatchedStatus for this **DataReader** (p. 976).*
- **rti::core::status::DataReaderCacheStatus** **datareader_cache_status** () const

  *<<**extension**>> (p. 354) Get the DataReaderCacheStatus for this **DataReader** (p. 976)*
- **rti::core::status::DataReaderProtocolStatus** **datareader_protocol_status** () const

  *<<**extension**>> (p. 354) Get the DataReaderProtocolStatus for this **DataReader** (p. 976)*
- **rti::core::status::DataReaderProtocolStatus** **matched_publication_datareader_protocol_status** (const **dds::core::InstanceHandle** &publication_handle)

  *<<**extension**>> (p. 354) Get the DataReaderProtocolStatus for this **DataReader** (p. 976)*
- void **acknowledge_all** ()

  *<<**extension**>> (p. 354) Acknowledge all previously accessed samples*
- void **acknowledge_all** (const AckResponseData &response_data)

  *<<**extension**>> (p. 354) Acknowledge all previously accessed samples*
- void **acknowledge_sample** (const **dds::sub::SampleInfo** &sample_info)

  *<<**extension**>> (p. 354) Acknowledge a single sample explicitly*
- void **acknowledge_sample** (const **dds::sub::SampleInfo** &sample_info, const AckResponseData &response↩ _data)

  *<<**extension**>> (p. 354) Acknowledge a single sample explicitly*
- void **close_contained_entities** ()

  *<<**extension**>> (p. 354) Closes all the entities created from this **DataReader** (p. 976)*
- **rti::core::Result**< **dds::sub::LoanedSamples**< T > > **read_noexcept** () noexcept

  *<<**extension**>> (p. 354) Alternative to **read()** (p. 996) that doesn't throw exceptions.*
- **rti::core::Result**< **dds::sub::LoanedSamples**< T > > **take_noexcept** () noexcept

  *<<**extension**>> (p. 354) Alternative to **take()** (p. 997) that doesn't throw exceptions.*
- bool **read** (T &sample)

  *<<**extension**>> (p. 354) Copies the next not-previously-accessed valid data value from the **DataReader** (p. 976) via a read operation.*

- bool **read** (T &sample, **dds::sub::SampleInfo** &sample_info)

  <<***extension***>> (p. *354) Copies the next not-previously-accessed data value from the* ***DataReader*** (p. *976) via a read operation.*
- bool **take** (T &sample)

  <<***extension***>> (p. *354) Copies the next not-previously-accessed valid data value from the* ***DataReader*** (p. *976) via a take operation.*
- bool **take** (T &sample, **dds::sub::SampleInfo** &sample_info)

  <<***extension***>> (p. *354) Copies the next not-previously-accessed data value from the* ***DataReader*** (p. *976) via a take operation.*
- bool **is_data_consistent** (const T &sample, const **dds::sub::SampleInfo** &sample_info)

  <<***extension***>> (p. *354) Checks if the sample has been overwritten by the DataWriter*
- bool **is_data_consistent** (const **rti::sub::LoanedSample**< T > &sample)

  <<***extension***>> (p. *354) Checks if the sample has been overwritten by the DataWriter*
- const std::string & **topic_name** () const

  <<***extension***>> (p. *354) Get the topic name associated with this* ***DataReader*** (p. *976)*
- const std::string & **type_name** () const

  <<***extension***>> (p. *354) Get the type name associated with this* ***DataReader*** (p. *976)*
- void **close** ()

  Close this ***DataReader*** (p. *976).*

Related Functions

(Note that these are not member functions.)

- bool **read** (**dds::sub::ReadModeDummyType**)

  The stream manipulator to indicate that the reader should read samples as opposed to taking the samples.
- bool **take** (**dds::sub::ReadModeDummyType**)

  The stream manipulator to indicate that the reader should take samples as opposed to reading the samples.
- dds::sub::functors::MaxSamplesManipulatorFunctor **max_samples** (uint32_t n)

  Stream manipulator to set the maximum number of samples to read or take.
- dds::sub::functors::ContentFilterManipulatorFunctor **content** (const **dds::sub::Query** &query)

  Stream manipulator to set a ***Query*** (p. *2053) to use during the subsequent read/take operation.*
- dds::sub::functors::ConditionManipulatorFunctor **condition** (const **dds::sub::cond::ReadCondition** &condition)

  Stream manipulator to set a QueryCondition to use during the subsequent read/take operation.
- dds::sub::functors::StateFilterManipulatorFunctor **state** (const **dds::sub::status::DataState** &s)

  Stream manipulator to specify the DataState of the samples that should be read/taken.
- dds::sub::functors::InstanceManipulatorFunctor **instance** (const **dds::core::InstanceHandle** &h)

  Stream manipulator to specify the instance whose samples should be read or taken.
- dds::sub::functors::NextInstanceManipulatorFunctor **next_instance** (const **dds::core::InstanceHandle** &h)

  Stream manipulator to specify the samples belonging to the 'next' instance after the provided instance handle should be accessed.
- void **ignore** (**dds::domain::DomainParticipant** &participant, const **dds::core::InstanceHandle** &handle)

  Instructs RTI Connext to locally ignore a subscription.
- template<typename FwdIterator >
  void **ignore** (**dds::domain::DomainParticipant** &participant, FwdIterator **begin**, FwdIterator **end**)

  Instructs RTI Connext to locally ignore subscriptions.

- template<typename T >
  const **::dds::core::InstanceHandleSeq matched_publications** (const **dds::sub::DataReader**< T > &reader)

  *Retrieve the list of publications currently "associated" with a **DataReader** (p. 976).*

- template<typename T , typename FwdIterator >
  FwdIterator **matched_publications** (const **dds::sub::DataReader**< T > &reader, FwdIterator **begin**, FwdIterator **end**)

  *Retrieve the list of publications currently "associated" with a **DataReader** (p. 976).*

- template<typename T >
  const **dds::topic::PublicationBuiltinTopicData matched_publication_data** (const **dds::sub::DataReader**< T > &reader, const **dds::core::InstanceHandle** &handle)

  *This operation retrieves the information on a publication that is currently "associated" with the **DataReader** (p. 976).*

- template<typename T >
  dds::topic::ParticipantBuiltinTopicData matched_publication_participant_data (const **dds::sub::DataReader**< T > &reader, const **dds::core::InstanceHandle** &handle)

  <<***extension***>> *(p. 354) This operation retrieves the information on the discovered **dds::domain::DomainParticipant** (p. 1326) associated with the publication that is currently matching with the **dds::sub::DataReader** (p. 976).*

- template<typename T >
  bool **is_matched_publication_alive** (const **dds::sub::DataReader**< T > &reader, const **dds::core::InstanceHandle** &handle)

  <<***extension***>> *(p. 354) Check if a matched publication is alive.*

- template<typename T >
  std::vector< **dds::topic::PublicationBuiltinTopicData** > **matched_publication_data** (const **dds::sub::DataReader**< T > &reader)

  <<***extension***>> *(p. 354) Obtain the PublicationBuiltinTopicData for all of the publications matched with a **DataReader** (p. 976).*

- template<typename Reader , typename FwdIterator >
  uint32_t **find** (const **dds::sub::Subscriber** & **subscriber**, RTI_STRING_IN **topic_name**, FwdIterator **begin**, uint32_t max_size)

  *This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching topic name.*

- template<typename Reader , typename BinIterator >
  uint32_t **find** (const **dds::sub::Subscriber** & **subscriber**, RTI_STRING_IN **topic_name**, BinIterator **begin**)

  *This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching topic name.*

- template<typename READER , typename T , typename FwdIterator >
  uint32_t **find** (const **dds::sub::Subscriber** & **subscriber**, const **dds::topic::TopicDescription**< T > & **topic_description**, FwdIterator **begin**, uint32_t max_size)

- template<typename READER , typename T , typename BinIterator >
  uint32_t **find** (const **dds::sub::Subscriber** & **subscriber**, const **dds::topic::TopicDescription**< T > & **topic_description**, BinIterator **begin**)

  *This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching TopicDescription.*

- template<typename AnyDataReaderBackInsertIterator >
  uint32_t **find_datareaders** (**dds::sub::Subscriber subscriber**, AnyDataReaderBackInsertIterator **begin**)

  <<***extension***>> *(p. 354) Retrieve all the **dds::sub::DataReader** (p. 976) created from this **dds::sub::Subscriber** (p. 2410)*

- template<typename AnyDataReaderForwardIterator >
  uint32_t **find_datareaders** (**dds::sub::Subscriber subscriber**, AnyDataReaderForwardIterator **begin**, uint32_t max_size)

  <<***extension***>> *(p. 354) Retrieve all the readers created from a subscriber.*

- template<typename Reader >
  Reader **find_datareader_by_topic_name** (**dds::sub::Subscriber subscriber**, const std::string & **topic_name**)

  <<***extension***>> *(p. 354) Retrieves a* ***dds::sub::DataReader*** *(p. 976) with the given topic name within the* ***dds::sub::Subscriber*** *(p. 2410)*
- template<typename Reader >
  Reader **find_datareader_by_name** (**dds::sub::Subscriber subscriber**, const std::string &datareader_name)

  <<***extension***>> *(p. 354) Retrieves a* ***dds::sub::DataReader*** *(p. 976) with the given name within the* ***dds::sub::Subscriber*** *(p. 2410)*
- template<typename Reader , typename T >
  Reader **find_datareader_by_topic_description** (const **dds::sub::Subscriber** & **subscriber**, const **dds::topic::TopicDescription**< T > & **topic_description**)

  <<***extension***>> *(p. 354) Retrieves a* ***dds::sub::DataReader*** *(p. 976) with the given TopicDescription within the* ***dds::sub::Subscriber*** *(p. 2410)*
- template<typename Reader >
  Reader **find_datareader_by_name** (**dds::domain::DomainParticipant** participant, const std::string &datareader_name)

  <<***extension***>> *(p. 354) Retrieves a* ***dds::sub::DataReader*** *(p. 976) within the* ***dds::domain::DomainParticipant*** *(p. 1326) with the given name*

8.53.1 Detailed Description

template<typename T>
class dds::sub::DataReader< T >

<<***reference-type***>> *(p. 351)* Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **dds::sub::Subscriber** (p. 2410).

Template Parameters

T	The topic-type that the **DataReader** (p. 976) subscribes to

QoS:

dds::sub::qos::DataReaderQos (p. 1072)

Status:

dds::core::status::StatusMask::data_available() (p. 2383);
dds::core::status::StatusMask::liveliness_changed() (p. 2383), **dds::core::status::LivelinessChangedStatus** (p. 1652);
dds::core::status::StatusMask::requested_deadline_missed() (p. 2380), **dds::core::status::RequestedDeadlineMissedStatus** (p. 2184);
dds::core::status::StatusMask::requested_incompatible_qos() (p. 2381), **dds::core::status::RequestedIncompatibleQosStatus** (p. 2185);
dds::core::status::StatusMask::sample_lost() (p. 2381), **dds::core::status::SampleLostStatus** (p. 2303);
dds::core::status::StatusMask::sample_rejected() (p. 2382), **dds::core::status::SampleRejectedStatus** (p. 2315);

dds::core::status::StatusMask::subscription_matched() (p. 2384), **dds::core::status::Subscription↩ MatchedStatus** (p. 2439);

Listener:

dds::sub::DataReaderListener (p. 1055)

A **dds::sub::DataReader** (p. 976) refers to exactly one TopicDescription (either a **dds::topic::Topic** (p. 2474), a **dds↩ ::topic::ContentFilteredTopic** (p. 954) or a MultiTopic) that identifies the data to be read.

The subscription has a unique resulting type. The data-reader may give access to several instances of the resulting type, which can be distinguished from each other by their `key`.

The following operations may be called even if the **dds::sub::DataReader** (p. 976) is not enabled. Other operations will fail with the value **dds::core::NotEnabledError** (p. 1869) if called on a disabled **dds::sub::DataReader** (p. 976):

- **dds::sub::DataReader::qos(const dds::sub::qos::DataReaderQos&)** (p. 1008), **dds::sub::DataReader↩ ::qos() const** (p. 1007), **dds::sub::DataReader::set_listener** (p. 1006), **dds::sub::DataReader::get↩ _listener** (p. 1007), **dds::core::Entity::enable** (p. 1524), **dds::core::cond::StatusCondition::Status↩ Condition(const dds::core::Entity&)** (p. 2373), **dds::core::Entity::status_changes()** (p. 1525), **dds::sub::↩ DataReader::liveliness_changed_status** (p. 1011), **dds::sub::DataReader::requested_deadline_missed↩ _status** (p. 1011), **dds::sub::DataReader::requested_incompatible_qos_status** (p. 1012), **dds::sub::Data↩ Reader::sample_lost_status** (p. 1011), **dds::sub::DataReader::sample_rejected_status** (p. 1011), **dds↩ ::sub::DataReader::subscription_matched_status** (p. 1012)

All sample-accessing operations may fail with the expection **dds::core::PreconditionNotMetError** (p. 1941) as described in **dds::sub::CoherentAccess::CoherentAccess()** (p. 930).

See also

Operations Allowed in Listener Callbacks (p. 1639)

Access to data samples (p. 62)

DataReader Use Cases (p. 316)

Entity Use Cases (p. 325)

Examples

Foo_subscriber.cxx.

8.53.2 Constructor & Destructor Documentation

8.53.2.1 DataReader() [1/12]

```
template<typename T >
dds::sub::DataReader< T >::DataReader (
            const  dds::sub::Subscriber & sub,
            const  dds::topic::Topic< T > & topic )  [inline]
```

Create a **DataReader** (p. 976).

It uses the default DataReaderQos, and no listener.

Parameters

sub	the **Subscriber** (p. 2410) owning this **DataReader** (p. 976).
topic	the Topic associated with this **DataReader** (p. 976).

8.53.2.2 DataReader() [2/12]

```
template<typename T >
dds::sub::DataReader< T >::DataReader (
             const  dds::sub::Subscriber & sub,
             const  dds::topic::Topic< T > & topic,
             const  dds::sub::qos::DataReaderQos & the_qos,
              dds::sub::DataReaderListener< T > * the_listener = NULL,
             const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a **DataReader** (p. 976).

[DEPRECATED] When using a listener, prefer the constructor that receives a shared_ptr<**Listener** (p. 1637)> instead of a regular Listener* pointer.

When a **DataReader** (p. 976) is created, only those transports already registered are available to the **DataReader** (p. 976). See **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered.

MT Safety:

UNSAFE. If the qos parameter is omitted it is not safe to create the datareader while another thread may be simultaneously calling dds::sub::Subscriber::default_datareader_qos(dds::sub::qos::DataReaderQos).

Precondition

If sub is enabled, the topic must be enabled. If it is not, this operation will fail and no **DataReader** (p. 976) will be created.

The given **dds::topic::Topic** (p. 2474) must have been created from the same participant as this subscriber. If it was created from a different participant no **DataReader** (p. 976) will be created.

Parameters

sub	the **Subscriber** (p. 2410) owning this **DataReader** (p. 976).
topic	the Topic associated with this **DataReader** (p. 976).
the_qos	the QoS settings for this **DataReader** (p. 976).
the_listener	the listener.
mask	the event mask associated to the **DataReader** (p. 976) listener.

See also

Specifying QoS on entities (p. 513) for information on setting QoS before entity creation

dds::sub::qos::DataReaderQos (p. 1072) for rules on consistency among QoS

dds::core::QosProvider::datareader_qos() (p. 2033)

dds::sub::qos::DataReaderQos::operator=(const dds::topic::qos::TopicQos&) (p. 1076) which allows assigning the contents of a TopicQos into a DataReaderQos

listener() (p. 1006)

8.53.2.3 DataReader() [3/12]

```
template<typename T >
dds::sub::DataReader< T >::DataReader (
             const  dds::sub::Subscriber & sub,
             const  dds::topic::Topic< T > & topic,
             const  dds::sub::qos::DataReaderQos & the_qos,
             std::shared_ptr<  Listener > the_listener,
             const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a **DataReader** (p. 976).

When a **DataReader** (p. 976) is created, only those transports already registered are available to the **DataReader** (p. 976). See **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered.

MT Safety:

UNSAFE. If the qos parameter is omitted it is not safe to create the datareader while another thread may be simultaneously calling dds::sub::Subscriber::default_datareader_qos(dds::sub::qos::DataReaderQos).

Precondition

If sub is enabled, the topic must be enabled. If it is not, this operation will fail and no **DataReader** (p. 976) will be created.

The given **dds::topic::Topic** (p. 2474) must have been created from the same participant as this subscriber. If it was created from a different participant no **DataReader** (p. 976) will be created.

Parameters

sub	the **Subscriber** (p. 2410) owning this **DataReader** (p. 976).
topic	the Topic associated with this **DataReader** (p. 976).
the_qos	the QoS settings for this **DataReader** (p. 976).
the_listener	A shared_ptr to the listener. See **set_listener()** (p. 1006) for more information.
mask	Indicates which status updates the listener will receive

See also

Specifying QoS on entities (p. 513) for information on setting QoS before entity creation

dds::sub::qos::DataReaderQos (p. 1072) for rules on consistency among QoS

dds::core::QosProvider::datareader_qos() (p. 2033)

dds::sub::qos::DataReaderQos::operator=(const dds::topic::qos::TopicQos&) (p. 1076) which allows assigning the contents of a TopicQos into a DataReaderQos

8.53.2.4 DataReader() [4/12]

```
template<typename T >
dds::sub::DataReader< T >::DataReader (
            const  dds::sub::Subscriber & sub,
            const  dds::topic::ContentFilteredTopic< T > & topic )  [inline]
```

Create a **DataReader** (p. 976) for a **dds::topic::ContentFilteredTopic** (p. 954).

This **DataReader** (p. 976) will only receive that data that matches the Filter associated with the ContentFilteredTopic. The QoS will be set to sub.default_datareader_qos().

Parameters

sub	the **Subscriber** (p. 2410) owning this **DataReader** (p. 976).
topic	the content filtered topic

8.53.2.5 DataReader() [5/12]

```
template<typename T >
dds::sub::DataReader< T >::DataReader (
            const  dds::sub::Subscriber & sub,
            const  dds::topic::ContentFilteredTopic< T > & topic,
            const  dds::sub::qos::DataReaderQos & the_qos,
             dds::sub::DataReaderListener< T > * the_listener = NULL,
            const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a **DataReader** (p. 976) for a **dds::topic::ContentFilteredTopic** (p. 954).

[DEPRECATED] When using a listener, prefer the constructor that receives a shared_ptr<**Listener** (p. 1637)> instead of a regular Listener* pointer.

This **DataReader** (p. 976) will only receive that data that mathes the Filter associated with the ContentFilteredTopic.

Parameters

sub	the subscriber owning this **DataReader** (p. 976).
topic	the content filtered topic.
the_qos	the QoS settings for this **DataReader** (p. 976).
the_listener	the listener.
mask	the event mask associated to the **DataReader** (p. 976) listener.

8.53.2.6 DataReader() [6/12]

```
template<typename T >
dds::sub::DataReader< T >::DataReader (
            const  dds::sub::Subscriber & sub,
            const  dds::topic::ContentFilteredTopic< T > & topic,
            const  dds::sub::qos::DataReaderQos & the_qos,
            std::shared_ptr<  Listener > the_listener,
            const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a **DataReader** (p. 976) for a **dds::topic::ContentFilteredTopic** (p. 954) with a listener.

This **DataReader** (p. 976) will only receive that data that mathes the Filter associated with the ContentFilteredTopic.

Parameters

sub	the subscriber owning this **DataReader** (p. 976).
topic	the content filtered topic.
the_qos	the QoS settings for this **DataReader** (p. 976).
the_listener	the listener.
mask	the event mask associated to the **DataReader** (p. 976) listener.

8.53.2.7 DataReader() [7/12]

```
template<typename T >
dds::sub::DataReader< T >::DataReader (
            const  dds::topic::Topic< T > & topic )  [inline], [explicit]
```

Create a **DataReader** (p. 976).

It uses default DataReaderQos, and no listener. The reader is created in the implicit subscriber (**rti::sub::implicit_subscriber** (p. 754)(topic.participant()))

Parameters

topic	the Topic associated with this **DataReader** (p. 976).

8.53.2.8 DataReader() [8/12]

```
template<typename T >
dds::sub::DataReader< T >::DataReader (
            const  dds::topic::Topic< T > & topic,
            const  dds::sub::qos::DataReaderQos & the_qos )  [inline]
```

Create a **DataReader** (p. 976) with QoS.

The reader is created in the implicit subscriber.

Parameters

topic	the Topic associated with this **DataReader** (p. 976).
the_qos	the QoS settings for this **DataReader** (p. 976).

8.53.2.9 DataReader() [9/12]

```
template<typename T >
dds::sub::DataReader< T >::DataReader (
            const  dds::topic::ContentFilteredTopic< T > & topic,
            const  dds::sub::qos::DataReaderQos & the_qos )  [inline]
```

Create a **DataReader** (p. 976) for a **dds::topic::ContentFilteredTopic** (p. 954).

This **DataReader** (p. 976) will only receive that data that matches the Filter associated with the ContentFilteredTopic.

The reader is created in the implicit subscriber.

Parameters

topic	the content filtered topic.
the_qos	the QoS settings for this **DataReader** (p. 976).

8.53.2.10 DataReader() [10/12]

```
template<typename T >
dds::sub::DataReader< T >::DataReader (
            const  dds::topic::Topic< T > & topic,
            const  dds::sub::qos::DataReaderQos & the_qos,
            std::shared_ptr<  Listener > the_listener,
            const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a **DataReader** (p. 976).

When a **DataReader** (p. 976) is created, only those transports already registered are available to the **DataReader** (p. 976). See **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered.

MT Safety:

UNSAFE. If the qos parameter is omitted it is not safe to create the datareader while another thread may be simultaneously calling dds::sub::Subscriber::default_datareader_qos(dds::sub::qos::DataReaderQos).

The reader is created in the implicit subscriber (**rti::sub::implicit_subscriber** (p. 754)(topic.↩ participant()))

Parameters

topic	the Topic associated with this **DataReader** (p. 976).
the_qos	the QoS settings for this **DataReader** (p. 976).
the_listener	A shared_ptr to the listener. See **set_listener()** (p. 1006) for more information.
mask	Indicates which status updates the listener will receive

See also

Specifying QoS on entities (p. 513) for information on setting QoS before entity creation

dds::sub::qos::DataReaderQos (p. 1072) for rules on consistency among QoS

dds::core::QosProvider::datareader_qos() (p. 2033)

dds::sub::qos::DataReaderQos::operator=(const dds::topic::qos::TopicQos&) (p. 1076) which allows assigning the contents of a TopicQos into a DataReaderQos

8.53.2.11 DataReader() [11/12]

```
template<typename T >
dds::sub::DataReader< T >::DataReader (
            const  dds::topic::ContentFilteredTopic< T > & topic )  [inline], [explicit]
```

Create a **DataReader** (p. 976) for a **dds::topic::ContentFilteredTopic** (p. 954).

This **DataReader** (p. 976) will only receive that data that matches the Filter associated with the ContentFilteredTopic. The QoS will be set to sub.default_datareader_qos().

The reader is created in the implicit subscriber.

Parameters

topic	the content filtered topic

8.53.2.12 DataReader() [12/12]

```
template<typename T >
dds::sub::DataReader< T >::DataReader (
            const  dds::topic::ContentFilteredTopic< T > & topic,
            const  dds::sub::qos::DataReaderQos & the_qos,
            std::shared_ptr<  Listener > the_listener,
            const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a **DataReader** (p. 976) for a **dds::topic::ContentFilteredTopic** (p. 954) with a listener.

This **DataReader** (p. 976) will only receive that data that matches the Filter associated with the ContentFilteredTopic.

The reader is created in the implicit subscriber.

Parameters

topic	the content filtered topic.
the_qos	the QoS settings for this **DataReader** (p. 976).
the_listener	the listener.
mask	the event mask associated to the **DataReader** (p. 976) listener.

8.53.3 Member Function Documentation

8.53.3.1 default_filter_state() [1/2]

```
template<typename T >
const  dds::sub::status::DataState &  dds::sub::DataReader< T >::default_filter_state ( )  [inline]
```

Returns the default state for read/take operations.

If left as default, it is set to **dds::sub::status::DataState::any()** (p. 1124).

8.53.3.2 default_filter_state() [2/2]

```
template<typename T >
DataReader &  dds::sub::DataReader< T >::default_filter_state (
             const  dds::sub::status::DataState & state )  [inline]
```

Set the default state filter for read/take operations.

Parameters

state	the state mask that will be used to read/take samples.

8.53.3.3 operator>>() [1/4]

```
template<typename T >
DataReader &  dds::sub::DataReader< T >::operator>> (
              dds::sub::LoanedSamples< T > & ls )  [inline]
```

Use the stream operator to take samples, placing them into a **LoanedSamples** (p. 1663) container.

For example:
```
reader » loaned_samples;
```

If you want to read samples, instead of take, you must use the stream manipulator **read(dds::sub::ReadMode↩ DummyType)** (p. 1024) and operator>>(bool(∗manipulator)(**ReadModeDummyType** (p. 2144)))

For example:
```
reader » read » loaned_samples;
```

Parameters

ls	The **LoanedSamples** (p. 1663) container to put the taken samples into

See also

operator>>(bool(∗manipulator)(**ReadModeDummyType** (p. 2144)))

operator>>(Functor f) (p. 995)

Accessing Received Data (p. 318)

8.53.3.4 operator>>() [2/4]

```
template<typename T >
ManipulatorSelector  dds::sub::DataReader< T >::operator>> (
            bool(*)( ReadModeDummyType) manipulator )  [inline]
```

Use the stream operator to read or take samples.

The read and take manipulators are defined externally to make it possible to control whether the stream operators reads or takes.

This stream operator can be chained together with **operator>>(Functor f)** (p. 995) to set other various properties of the read/take operation.

It is used as follows:

```
dr » read » loanedSamples;
dr » take » loanedSamples;
```

If either the read or take manipulator is not passed into the stream, the default behavior is to take.

Parameters

manipulator	Either read or take

See also

operator>>(Functor f) (p. 995)

operator>>(dds::sub::LoanedSamples<T>& ls) (p. 993)

ReadModeDummyType (p. 2144)

Accessing Received Data (p. 318)

8.53.3.5 operator>>() [3/4]

```
template<typename T >
template<typename Functor >
ManipulatorSelector  dds::sub::DataReader< T >::operator>> (
            Functor f )  [inline]
```

Use the stream operator to dictate the behavior of the read or take operation.

There are a number of pre-defined stream manipulators (functions which return a functor that will be used by the stream operator). They dictate how the samples will be read or taken:

- max_samples: Read/Take up to a maximum number of samples
- content: Read/Take using a **Query** (p. 2053)
- condition: Read/Take using a **dds::sub::cond::ReadCondition** (p. 2133)
- state: Read/Take only samples with a specific DataState
- instance: Read/Take a specific instance
- next_instance: Read/Take the next instance after a specified instance

For example, to read up to 10 samples that have not been read before:

```
dr » read
   » max_samples(10)
   » state(DataState::new_data())
   » loanedSamples;
```

Parameters

f	Each of the stream manipulators listed above return a Functor that will be passed to the stream operator. These Functors do not need to be accessed by the user directly. They should instead be passed indirectly via the return value of one of the provided stream manipulators.

See also

dds::sub::max_samples(uint32_t n) (p. 662)

dds::sub::content(const dds::sub::Query& query) (p. 663)

dds::sub::condition(const dds::sub::cond::ReadCondition& query_condition) (p. 663)

dds::sub::state(const dds::sub::status::DataState& s) (p. 664)

dds::sub::instance(const dds::core::InstanceHandle& h) (p. 665)

dds::sub::next_instance(const dds::core::InstanceHandle& h) (p. 665)

Accessing Received Data (p. 318)

8.53.3.6 read() [1/5]

```
template<typename T >
LoanedSamples< T >  dds::sub::DataReader< T >::read ( )  [inline]
```

Read all samples using the default filter state.

This operation offers the same functionality and API as **dds::sub::DataReader::take** (p. 997) except that the samples returned remain in the **dds::sub::DataReader** (p. 976) such that they can be retrieved again by means of a read or take operation.

Please refer to the documentation of **dds::sub::DataReader::take** (p. 997) for details on the number of samples returned as well as the order in which the samples appear in these sequences.

The act of reading a sample changes its `sample_state` to **dds::sub::status::SampleState::read()** (p. 2318). If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the instance to be **dds::sub::status::ViewState::not_new_view()** (p. 2616). It will not affect the `instance_state` of the instance.

Once the application completes its use of the samples, it must 'return the loan' to the **dds::sub::DataReader** (p. 976). This can be done explicitly by calling the **dds::sub::LoanedSamples::return_loan()** (p. 1667) operation or implicitly by the **dds::sub::LoanedSamples** (p. 1663) destructor.

Important: When you loan data from the middleware, you *must not* keep any pointers to any part of the data samples or the **dds::sub::SampleInfo** (p. 2285) objects after the call to **dds::sub::LoanedSamples::return_loan()** (p. 1667). Returning the loan places the objects back into a pool, allowing the middleware to overwrite them with new data.

Note: While the loan must be returned at some point, you do *not* have to do so before the next **dds::sub::DataReader::take** (p. 997) call. However, failure to return the loan will eventually deplete the **dds::sub::DataReader** (p. 976) of the buffers it needs to receive new samples and eventually samples will start to be lost. The total number of buffers available to the **dds::sub::DataReader** (p. 976) is specified by the **dds::core::policy::ResourceLimits** (p. 2203) and the **rti::core::policy::DataReaderResourceLimits** (p. 1081).

Important: If the samples "returned" by this method are loaned from RTI Connext (see **dds::sub::DataReader::take** (p. 997) for more information on memory loaning), it is important that their contents not be changed. Because the memory in which the data is stored belongs to the middleware, any modifications made to the data will be seen the next time the same samples are read or taken; the samples will no longer reflect the state that was received from the network.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::PreconditionNotMetError** (p. 1941), **dds::core::NotEnabledError** (p. 1869).

See also

dds::sub::DataReader::take (p. 997)

Accessing Received Data (p. 318)

select() (p. 1003)

8.53.3.7 take() [1/5]

```
template<typename T >
LoanedSamples< T >  dds::sub::DataReader< T >::take ( )  [inline]
```

Take all samples using the default filter state.

The operation will return the list of samples received by the **dds::sub::DataReader** (p. 976) since the last **dds::sub::DataReader::take** (p. 997) operation that matches the specified SampleStateMask, ViewStateMask and InstanceStateMask.

This operation may fail with **dds::core::Error** (p. 1539) if the **rti::core::policy::DataReaderResourceLimits::max_outstanding_reads** (p. 1091) limit has been exceeded.

The actual number of samples returned depends on the information that has been received by the middleware as well as the **dds::core::policy::History** (p. 1602), **dds::core::policy::ResourceLimits** (p. 2203), **rti::core::policy::DataReaderResourceLimits** (p. 1081) and the characteristics of the data-type that is associated with the **dds::sub::DataReader** (p. 976):

- In the case where the **dds::core::policy::History::kind** (p. 1604) is **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607), the call will return at most **dds::core::policy::History::depth** (p. 1605) samples for each ALIVE instance and (**dds::core::policy::History::depth** (p. 1605) +1) samples for each NOT_ALIVE instance. The extra sample is an invalid sample (**dds::sub::SampleInfo::valid** (p. 2289) is FALSE) that is used to indicate the instance state transition from ALIVE to NOT_ALIVE.

- The maximum number of samples returned is limited by **dds::core::policy::ResourceLimits::max_samples** (p. 2205), and by **rti::core::policy::DataReaderResourceLimits::max_samples_per_read** (p. 1091).

- For multiple instances, the number of samples returned is additionally limited by the product (**dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207)
  - **dds::core::policy::ResourceLimits::max_instances** (p. 2206))

- If **rti::core::policy::DataReaderResourceLimits::max_infos** (p. 1087) is limited, the number of samples returned may also be limited if insufficient **dds::sub::SampleInfo** (p. 2285) resources are available.

If the read or take succeeds and the number of samples returned has been limited (by means of a maximum limit, as listed above, or insufficient **dds::sub::SampleInfo** (p. 2285) resources), the call will complete successfully and provide those samples the reader is able to return. The user may need to make additional calls, or return outstanding loaned buffers in the case of insufficient resources, in order to access remaining samples.

Note that in the case where the **dds::topic::Topic** (p. 2474) associated with the **dds::sub::DataReader** (p. 976) is bound to a data-type that has no key definition, then there will be at most one instance in the **dds::sub::DataReader** (p. 976). So the per-sample limits will apply.

The act of *taking* a sample removes it from RTI Connext so it cannot be read or taken again. If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the sample's instance to **dds::sub::status::ViewState::not_new_view()** (p. 2616). It will not affect the `instance_state` of the sample's instance.

Once the application completes its use of the samples it must 'return the loan' to the **dds::sub::DataReader** (p. 976) by calling the **dds::sub::LoanedSamples::return_loan()** (p. 1667) operation. The **dds::sub::LoanedSamples** (p. 1663) destructor also takes care of returning the loan for you, so calling **dds::sub::LoanedSamples::return_loan()** (p. 1667) explicitly is not required.

Important: When you loan data from the middleware, you *must not* keep any pointers to any part of the data samples or the **dds::sub::SampleInfo** (p. 2285) objects after the call to **dds::sub::LoanedSamples::return_loan()** (p. 1667). Returning the loan places the objects back into a pool, allowing the middleware to overwrite them with new data.

Note: While you must call **dds::sub::LoanedSamples::return_loan()** (p. 1667) at some point, you do *not* have to do so before the next **dds::sub::DataReader::take** (p. 997) call. However, failure to return the loan will eventually deplete the **dds::sub::DataReader** (p. 976) of the buffers it needs to receive new samples and eventually samples will start to be lost. The total number of buffers available to the **dds::sub::DataReader** (p. 976) is specified by the **dds::core::policy::ResourceLimits** (p. 2203) and the **rti::core::policy::DataReaderResourceLimits** (p. 1081).

To copy samples from the reader instead of taking a loan, you can use any of the read or take operations that receive an iterator to a container in which to copy the samples:

- **read(SamplesFWIterator sfit, int32_t max_samples)** (p. 1000)
- **take(SamplesFWIterator sfit, int32_t max_samples)** (p. 1001)
- **read(SamplesBIIterator sbit)** (p. 1001)
- **take(SamplesBIIterator sbit)** (p. 1002)

The order of the samples returned to the caller depends on the **dds::core::policy::Presentation** (p. 1942).

- If **dds::core::policy::Presentation::access_scope** (p. 1947) is **dds::core::policy::PresentationAccessScopeKind_def::INSTANCE** (p. 1951), the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **dds::core::policy::Presentation::access_scope** (p. 1947) is **dds::core::policy::PresentationAccessScopeKind_def::TOPIC** (p. 1951) and **dds::core::policy::Presentation::ordered_access** (p. 1948) is set to false, then the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **dds::core::policy::Presentation::access_scope** (p. 1947) is **dds::core::policy::PresentationAccessScopeKind_def::TOPIC** (p. 1951) and **dds::core::policy::Presentation::ordered_access** (p. 1948) is set to true, then the returned collection is a list where the relative order of samples as they were written by a DataWriter is preserved also across different instances. In other words, changes made by a single DataWriter are made available to subscribers in the same order in which they occur. Samples belonging to the same instance may or may not be consecutive. This is because, to preserve order within a single DataWriter, it may be necessary to mix samples from different instances.

- If **dds::core::policy::Presentation::access_scope** (p. 1947) is **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951) and **dds::core::policy::Presentation::ordered_access** (p. 1948) is set to false, then the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **dds::core::policy::Presentation::access_scope** (p. 1947) is **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951) and **dds::core::policy::Presentation::ordered_access** (p. 1948) is set to true, then changes made to instances by a set of DataWriters attached to a common Publisher are made available in the order in which they were written. For this to happen, the application must take the samples using the **Subscriber** (p. 2410)'s **dds::sub::CoherentAccess::CoherentAccess()** (p. 930) and **dds::sub::CoherentAccess::end()** (p. 931) operations.

See the "Ordered Access" section in the "PRESENTATION QosPolicy" section of the Core Libraries User's Manual for more information.

In all of the above cases, the relative order between the samples of one instance is consistent with the **DESTINATION_ORDER** (p. 524) policy:

- If **dds::core::policy::DestinationOrder::kind** (p. 1261) is **dds::core::policy::DestinationOrderKind_def::BY_RECEPTION_TIMESTAMP** (p. 1264), samples belonging to the same instances will appear in the relative order in which there were received (FIFO, earlier samples ahead of the later samples).

- If **dds::core::policy::DestinationOrder::kind** (p. 1261) is **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264), samples belonging to the same instances will appear in the relative order implied by the `source_timestamp` (FIFO, smaller values of `source_timestamp` ahead of the larger values).

In addition to the collection of samples, the read and take operations also use a collection of **dds::sub::SampleInfo** (p. 2285) structures.

Some elements in the returned collection may not have valid data. If the instance_state in the **dds::sub::SampleInfo** (p. 2285) is **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) or **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617), then the last sample for that instance in the collection (that is, the one whose **dds::sub::SampleInfo** (p. 2285) has sample_rank==0) does not contain valid data.

Samples that contain no data do not count towards the limits imposed by the **dds::core::policy::ResourceLimits** (p. 2203). The act of reading/taking a sample sets its sample_state to **dds::sub::status::SampleState::read()** (p. 2318).

If the sample belongs to the most recent generation of the instance, it will also set the view_state of the instance to **dds::sub::status::ViewState::not_new_view()** (p. 2616). It will not affect the instance_state of the instance.

For an example on how `take` can be used, please refer to the **Accessing Received Data** (p. 318) "receive example".

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::PreconditionNotMetError** (p. 1941), **dds::core::NotEnabledError** (p. 1869).

See also

dds::sub::DataReader::read (p. 996)

Accessing Received Data (p. 318)

select() (p. 1003)

Examples

Foo_subscriber.cxx.

8.53.3.8 read() [2/5]

```
template<typename T >
template<typename SamplesFWIterator >
uint32_t  dds::sub::DataReader< T >::read (
             SamplesFWIterator sfit,
             int32_t max_samples )  [inline]
```

Read up to max_samples samples using the default filter state.

The samples are copied into the application provided container using the forward iterator parameter.

Template Parameters

SamplesFWIterator	A forward iterator whose value type is dds::sub::Sample<T>

Parameters

sfit	samples forward iterator
max_samples	the maximum number of samples to read

Returns

uint32_t the number of read samples.

See also

read() (p. 996)

Accessing Received Data (p. 318)

8.53.3.9 take() [2/5]

```
template<typename T >
template<typename SamplesFWIterator >
uint32_t  dds::sub::DataReader< T >::take (
             SamplesFWIterator sfit,
             int32_t max_samples )  [inline]
```

Take up to max_samples samples using the default filter state.

The samples are copied into the application provided container using the forward iterator parameter.

Template Parameters

SamplesFWIterator	A forward iterator whose value type is dds::sub::Sample<T>

Parameters

sfit	samples forward iterator.
max_samples	the maximum number of samples to take.

Returns

uint32_t The number of taken samples.

See also

take() (p. 997)

Accessing Received Data (p. 318)

8.53.3.10 read() [3/5]

```
template<typename T >
template<typename SamplesBIIterator >
uint32_t  dds::sub::DataReader< T >::read (
             SamplesBIIterator sbit )  [inline]
```

Read all samples available in the reader cache using the default filter state.

The samples are copied into the application provided container using a back-inserting iterator. Notice that as a consequence of using a back-inserting iterator, this operation may allocate memory to resize the underlying container.

Template Parameters

SamplesBIIterator	A back-inserting iterator whose value type is dds::sub::Sample<T>.

Parameters

sbit	samples back-inserting iterator.

Returns

uint32_t the number of read samples.

For example:

```
std::vector<Foo> samples;
// inserts copies of all available samples at the end of 'samples'
uint32_t count = reader.read(std::back_inserter(samples));
assert(count == samples.size());
```

Note

This function should not be confused with **read(T& sample)** (p. 1018), an extension function that reads a single sample and assigns it to the argument.

See also

read() (p. 996)

Accessing Received Data (p. 318)

8.53.3.11 take() [3/5]

```
template<typename T >
template<typename SamplesBIIterator >
uint32_t  dds::sub::DataReader< T >::take (
             SamplesBIIterator sbit )  [inline]
```

Take all samples available in the reader cache samples using the default filter state.

The samples are copied into the application provided container using a back-inserting iterator. Notice that as a consequence of using a back-inserting iterator, this operation may allocate memory to resize the underlying container.

Template Parameters

SamplesBIIterator	A back-inserting iterator whose value type is dds::sub::Sample<T>

Parameters

sbit	samples back-inserting iterator.

Returns

the number of taken samples.

Note

This function should not be confused with **take(T& sample)** (p. 1020), an extension function that takes a single sample and assigns it to the argument.

See also

take() (p. 997)

Accessing Received Data (p. 318)

8.53.3.12 select()

```
template<typename T >
Selector dds::sub::DataReader< T >::select ( ) [inline]
```

Get a **Selector** (p. 2320) to perform complex data selections, such as per-instance selection, content and status filtering.

The selector can be used as follows:

```
LoanedSamples<Foo> samples = reader.select()
    .instance(handle)
    .content(query)
    .state(state)
    .take();
```

This shows how samples can be taken by selecting a specific instance, then filtering by state and content.

Note that when the application wants to access all available samples it can simply call **DataReader::read()** (p. 996) or **DataReader::take()** (p. 997).

Returns

A **Selector** (p. 2320), typically used in-line to configure it and finally call **Selector::read()** (p. 2325) or **Selector::take()** (p. 2326).

See also

Selector (p. 2320), for the different **parameters** (p. 2063) that can be set to configure what samples to **read** (p. 996) or **take** (p. 997).

Selecting what samples to read (p. 319)

8.53.3.13 key_value() [1/2]

```
template<typename T >
T &  dds::sub::DataReader< T >::key_value (
             T & key_holder,
             const  dds::core::InstanceHandle & handle )  [inline]
```

Retrieve the instance key that corresponds to an instance handle.

Useful for keyed data types.

The operation will only fill the fields that form the key inside the `key_holder` instance.

For keyed data types, this operation may fail with **dds::core::InvalidArgumentError** (p. 1619) if the `handle` does not correspond to an existing data-object known to the **DataReader** (p. 976).

Parameters

key_holder	A user data type specific key holder of type T whose key fields are filled by this operation. If T has no key, this operation has no effect.
handle	The instance whose key is to be retrieved. If T has a key, `handle` must represent an existing instance of type T known to the **DataReader** (p. 976). Otherwise, this method will fail with **dds::core::InvalidArgumentError** (p. 1619). If T has a key and `handle` is **dds::core::InstanceHandle::nil()** (p. 1614), this method will fail with **dds::core::InvalidArgumentError** (p. 1619). If T has a key and `handle` represents an instance of another type or an instance of type T that has been unregistered, this method will fail with **dds::core::InvalidArgumentError** (p. 1619). If T has no key, this method has no effect.

8.53.3.14 key_value() [2/2]

```
template<typename T >
dds::topic::TopicInstance< T > &  dds::sub::DataReader< T >::key_value (
              dds::topic::TopicInstance< T > & key_holder,
             const  dds::core::InstanceHandle & handle )  [inline]
```

Retrieve the instance key that corresponds to an instance handle.

Useful for keyed data types.

The operation will only fill the fields that form the key inside the `key_holder` instance.

For keyed data types, this operation may fail with **dds::core::InvalidArgumentError** (p. 1619) if the `handle` does not correspond to an existing data-object known to the **DataReader** (p. 976).

Parameters

key_holder	A TopicInstance whose sample's key fields are filled by this operation. If T does not have a key, this method has no effect.

Parameters

handle	The instance whose key is to be retrieved. If T has a key, `handle` must represent an existing instance of type T known to the **DataReader** (p. 976). Otherwise, this method will fail with **dds::core::InvalidArgumentError** (p. 1619). If T has a key and `handle` is **dds::core::InstanceHandle::nil()** (p. 1614), this method will fail with **dds::core::InvalidArgumentError** (p. 1619). If T has a key and `handle` represents an instance of another type or an instance of type T that has been unregistered, this method will fail with **dds::core::InvalidArgumentError** (p. 1619). If T has no key, this method has no effect.

8.53.3.15 lookup_instance()

```
template<typename T >
dds::core::InstanceHandle  dds::sub::DataReader< T >::lookup_instance (
             const T & key_holder ) const  [inline]
```

Retrieve the InstanceHandle that corresponds to an instance key holder.

Useful for keyed data types.

This operation takes as a parameter an instance and returns a handle that can be used in subsequent operations that accept an instance handle as an argument. The instance parameter is only used for the purpose of examining the fields that define the key. If the instance is unknown to the **DataReader** (p. 976), or if for any other reason the Service is unable to provide an instance handle, the Service will return the special value **dds::core::InstanceHandle::nil()** (p. 1614).

Parameters

key_holder	<<***in***>> *(*p. *355)* a user data type specific key holder.

Returns

the instance handle associated with this instance. If `T` has no key, this method has no effect and returns **dds::core::InstanceHandle::nil()** (p. 1614)

8.53.3.16 topic_description()

```
template<typename T >
dds::topic::TopicDescription< DataType >  dds::sub::DataReader< T >::topic_description ( ) const
[inline]
```

Returns the **dds::topic::TopicDescription** (p. 2503) associated with the **DataReader** (p. 976).

Returns that same TopicDescription that was used to create the **dds::sub::DataReader** (p. 976).

Returns

TopicDescription associated with the **dds::sub::DataReader** (p. 976).

8.53.3.17 subscriber()

```
template<typename T >
const  dds::sub::Subscriber &  dds::sub::DataReader< T >::subscriber ( ) const  [inline]
```

Returns the **Subscriber** (p. 2410) to which the **DataReader** (p. 976) belongs.

8.53.3.18 listener() [1/2]

```
template<typename T >
void  dds::sub::DataReader< T >::listener (
             Listener * the_listener,
            const  dds::core::status::StatusMask & event_mask )  [inline]
```

Sets the listener associated with this reader.

[DEPRECATED] The use of **set_listener()** (p. 1006) is recommended. Unlike this function, set_listener receives a shared_ptr which simplifies the management of listener's lifecycle.

Parameters

the_listener	The **DataReaderListener** (p. 1055) to set
event_mask	The **dds::core::status::StatusMask** (p. 2375) associated with the listener

8.53.3.19 listener() [2/2]

```
template<typename T >
Listener *  dds::sub::DataReader< T >::listener ( ) const  [inline]
```

Returns the listener currently associated with this **DataReader** (p. 976).

[DEPRECATED] Prefer **get_listener()** (p. 1007) instead of this function.

If there's no listener it returns NULL.

8.53.3.20 set_listener() [1/2]

```
template<typename T >
void  dds::sub::DataReader< T >::set_listener (
            std::shared_ptr<  Listener > the_listener,
            const  dds::core::status::StatusMask & event_mask )  [inline]
```

Sets the listener associated with this reader.

The **DataReader** (p. 976) will hold a shared_ptr to the listener argument, ensuring that it is not deleted at least until this **DataReader** (p. 976) is deleted or the listener is reset with set_listener(nullptr).

Warning

It's recommended that the listener implementation doesn't hold a permanent reference to the reader. If it does, the application needs to manually reset the listener or manually close this **DataReader** (p. 976) to ensure that there is no cycle that prevents the destruction of these two objects.

Parameters

the_listener	A shared pointer to the listener to receive status updates or `nullptr` to reset the current listener and stop receiving status updates.
event_mask	A mask that indicates which status updates will be notified to the listener

8.53.3.21 set_listener() [2/2]

```
template<typename T >
void  dds::sub::DataReader< T >::set_listener (
            std::shared_ptr<  Listener > the_listener )  [inline]
```

Sets the listener associated with this reader.

If `the_listener` is not `nullptr`, this overload is equivalent to:
```
reader.set_listener(the_listener, dds::core::status::StatusMask::all());
```

If `the_listener` is `nullptr`, it is equivalent to:
```
reader.set_listener(nullptr, dds::core::status::StatusMask::none());
```

Parameters

the_listener	A shared pointer to the listener to receive status updates or `nullptr` to reset the current listener and stop receiving status updates.

8.53.3.22 get_listener()

```
template<typename T >
std::shared_ptr<  Listener >  dds::sub::DataReader< T >::get_listener ( ) const  [inline]
```

Returns the listener currently associated with this **DataReader** (p. 976).

Returns

The shared pointer to the current listener or `nullptr` if there is currently no listener associated to this entity.

8.53.3.23 qos() [1/2]

```
template<typename T >
const  dds::sub::qos::DataReaderQos  dds::sub::DataReader< T >::qos ( ) const  [inline]
```

Get the QoS associated with this reader.

8.53.3.24 qos() [2/2]

```
template<typename T >
void  dds::sub::DataReader< T >::qos (
            const  dds::sub::qos::DataReaderQos & the_qos )  [inline]
```

Set the QoS associated with this reader.

This operation modifies the QoS of the **DataReader** (p. 976).

The **dds::core::policy::UserData** (p. 2591), **dds::core::policy::Deadline** (p. 1255), **dds::core::policy::Latency↩Budget** (p. 1631), **dds::core::policy::TimeBasedFilter** (p. 2469), **dds::core::policy::ReaderDataLifecycle** (p. 2137) can be changed. The other policies are immutable.

Parameters

the_qos	The DataReaderQos to be set to. Policies must be consistent. Immutable policies cannot be changed after the **DataReader** (p. 976) is enabled. **dds::sub::Subscriber::default_datareader_qos()** (p. 2417) can be used to set the QoS of the **DataReader** (p. 976) to match the current default DataReaderQos set in the **Subscriber** (p. 2410).

Exceptions

dds::core::ImmutablePolicyError *(p. 1609),or*	**dds::core::InconsistentPolicyError** (p. 1610).,

See also

dds::sub::qos::DataReaderQos (p. 1072) for rules on consistency among QoS

set_qos (abstract) (p. 1522)

Operations Allowed in Listener Callbacks (p. 1639)

8.53.3.25 operator<<()

```
template<typename T >
DataReader &  dds::sub::DataReader< T >::operator<< (
            const  dds::sub::qos::DataReaderQos & the_qos )  [inline]
```

Set the QoS associated with this reader.

Parameters

the_qos	the new qos for this **DataReader** (p. 976).

See also

qos(const dds::sub::qos::DataReaderQos& qos) (p. 1008)

8.53.3.26 operator>>() [4/4]

```
template<typename T >
DataReader &  dds::sub::DataReader< T >::operator>> (
              dds::sub::qos::DataReaderQos & the_qos )  [inline]
```

Get the QoS associated with this reader.

Parameters

the_qos	the object to populate with the qos for this DataWriter.

See also

qos() (p. 1007)

8.53.3.27 wait_for_historical_data()

```
template<typename T >
void  dds::sub::DataReader< T >::wait_for_historical_data (
            const  dds::core::Duration & max_wait )  [inline]
```

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind.

This operation is intended only for **dds::sub::DataReader** (p. 976) entities that have a non-VOLATILE Durability QoS kind.

As soon as an application enables a non-VOLATILE **dds::sub::DataReader** (p. 976), it will start receiving both "historical" data (i.e., the data that was written prior to the time the **dds::sub::DataReader** (p. 976) joined the domain) as well as any new data written by the **dds::pub::DataWriter** (p. 1138) entities. There are situations where the application logic may require the application to wait until all "historical" data is received. This is the purpose of the **dds::sub::DataReader::wait_for_historical_data** (p. 1009) operations.

dds::sub::DataReader::wait_for_historical_data() (p. 1009) blocks the calling thread until either all "historical" data is received, or until the duration specified by the max_wait parameter elapses, whichever happens first. It will return

immediately if no DataWriters have been discovered at the time the operation is called; therefore it is advisable to make sure at least one **dds::pub::DataWriter** (p. 1138) has been discovered before calling this operation. (One way to do this is by using **dds::sub::DataReader::subscription_matched_status** (p. 1012).)

A successful completion indicates that all the "historical" data was "received"; timing out indicates that max_wait elapsed before all the data was received.

Parameters

max_wait	<<***in***>> *(*p. *355)* Timeout value.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::TimeoutError** (p. 2473) or **dds::core::NotEnabledError** (p. 1869).

8.53.3.28 liveliness_changed_status()

```
template<typename T >
dds::core::status::LivelinessChangedStatus dds::sub::DataReader< T >::liveliness_changed_status
( ) const [inline]
```

Get the LivelinessChangedStatus for this **DataReader** (p. 976).

See also

dds::core::status::LivelinessChangedStatus (p. 1652)

8.53.3.29 sample_rejected_status()

```
template<typename T >
dds::core::status::SampleRejectedStatus dds::sub::DataReader< T >::sample_rejected_status ( )
const [inline]
```

Get the SampleRejectedStatus for this **DataReader** (p. 976).

See also

dds::core::status::SampleRejectedStatus (p. 2315)

8.53.3.30 sample_lost_status()

```
template<typename T >
dds::core::status::SampleLostStatus dds::sub::DataReader< T >::sample_lost_status ( ) const
[inline]
```

Get the SampleLostStatus for this **DataReader** (p. 976).

See also

dds::core::status::SampleLostStatus (p. 2303)

8.53.3.31 requested_deadline_missed_status()

```
template<typename T >
dds::core::status::RequestedDeadlineMissedStatus dds::sub::DataReader< T >::requested_deadline↩
_missed_status ( ) [inline]
```

Get the RequestedDeadlineMissedStatus for this **DataReader** (p. 976).

See also

dds::core::status::RequestedDeadlineMissedStatus (p. 2184)

8.53.3.32 requested_incompatible_qos_status()

```
template<typename T >
dds::core::status::RequestedIncompatibleQosStatus dds::sub::DataReader< T >::requested_incompatible↩
_qos_status ( ) const [inline]
```

Get the RequestedIncompatibleQosStatus for this **DataReader** (p. 976).

See also

dds::core::status::RequestedIncompatibleQosStatus (p. 2185)

8.53.3.33 subscription_matched_status()

```
template<typename T >
dds::core::status::SubscriptionMatchedStatus dds::sub::DataReader< T >::subscription_matched_↩
status ( ) const [inline]
```

Get the SubscriptionMatchedStatus for this **DataReader** (p. 976).

See also

dds::core::status::SubscriptionMatchedStatus (p. 2439)

8.53.3.34 datareader_cache_status()

```
template<typename T >
rti::core::status::DataReaderCacheStatus datareader_cache_status ( ) const
```

<<***extension***>> *(p. 354)* Get the DataReaderCacheStatus for this **DataReader** (p. 976)

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

See also

rti::core::status::DataReaderCacheStatus (p. 1046)

8.53.3.35 datareader_protocol_status()

```
template<typename T >
rti::core::status::DataReaderProtocolStatus datareader_protocol_status ( ) const
```

<<***extension***>> *(p. 354)* Get the DataReaderProtocolStatus for this **DataReader** (p. 976)

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

See also

rti::core::status::DataReaderProtocolStatus (p. 1065)

8.53.3.36 matched_publication_datareader_protocol_status()

```
template<typename T >
rti::core::status::DataReaderProtocolStatus matched_publication_datareader_protocol_status (
            const  dds::core::InstanceHandle & publication_handle )
```

<<***extension***>> *(p. 354)* Get the DataReaderProtocolStatus for this **DataReader** (p. 976)

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

See also

rti::core::status::DataReaderProtocolStatus (p. 1065)

8.53.3.37 acknowledge_all() [1/2]

```
template<typename T >
void acknowledge_all ( )
```

<<***extension***>> *(*p. *354)* Acknowledge all previously accessed samples

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Applicable only when **dds::core::policy::Reliability::acknowledgment_kind** (p. 2154) = **rti::core::policy::AcknowledgmentKind_def::APPLICATION_EXPLICIT** (p. 798)

A call to this method does not necessarily trigger the sending of an AppAck RTPS message from the **DataReader** (p. 976) to the DataWriter. How and when AppAck messages are sent can be configured using the QoS values **rti::core::RtpsReliableReaderProtocol::samples_per_app_ack** (p. 2228) and **rti::core::RtpsReliableReaderProtocol::app_ack_period** (p. 2227).

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.53.3.38 acknowledge_all() [2/2]

```
template<typename T >
void acknowledge_all (
            const AckResponseData & response_data )
```

<<***extension***>> *(*p. *354)* Acknowledge all previously accessed samples

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Applicable only when **dds::core::policy::Reliability::acknowledgment_kind** (p. 2154) = **rti::core::policy::AcknowledgmentKind_def::APPLICATION_EXPLICIT** (p. 798)

A call to this method does not necessarily trigger the sending of an AppAck RTPS message from the **DataReader** (p. 976) to the DataWriter. How and when AppAck messages are sent can be configured using the QoS values **rti::core::RtpsReliableReaderProtocol::samples_per_app_ack** (p. 2228) and **rti::core::RtpsReliableReaderProtocol::app_ack_period** (p. 2227).

The maximum length of the response is configured using **rti::core::policy::DataReaderResourceLimits::max_app_ack_response_length** (p. 1101).

Parameters

response_data	<<***in***>> *(*p. *355)* Response data sent to **dds::pub::DataWriter** (p. 1138) upon acknowledgment

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.53.3.39 acknowledge_sample() [1/2]

```
template<typename T >
void acknowledge_sample (
            const  dds::sub::SampleInfo & sample_info )
```

<<***extension***>> *(*p. *354)* Acknowledge a single sample explicitly

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Applicable only when **dds::core::policy::Reliability::acknowledgment_kind** (p. 2154) = **rti::core::policy::↩ AcknowledgmentKind_def::APPLICATION_EXPLICIT** (p. 798)

A call to this method does not necessarily trigger the sending of an AppAck RTPS message from the **Data↩ Reader** (p. 976) to the DataWriter. How and when AppAck messages are sent can be configured using the QoS values **rti::core::RtpsReliableReaderProtocol::samples_per_app_ack** (p. 2228) and **rti::core::RtpsReliableReader↩ Protocol::app_ack_period** (p. 2227).

Parameters

sample_info	<<***in***>> *(*p. *355)* **dds::sub::SampleInfo** (p. 2285) identifying the sample being acknowledged.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.53.3.40 acknowledge_sample() [2/2]

```
template<typename T >
void acknowledge_sample (
```

```
const  dds::sub::SampleInfo & sample_info,
const AckResponseData & response_data )
```

<<***extension***>> *(p. 354)* Acknowledge a single sample explicitly

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Applicable only when **dds::core::policy::Reliability::acknowledgment_kind** (p. 2154) = **rti::core::policy::AcknowledgmentKind_def::APPLICATION_EXPLICIT** (p. 798)

A call to this method does not necessarily trigger the sending of an AppAck RTPS message from the **DataReader** (p. 976) to the DataWriter. How and when AppAck messages are sent can be configured using the QoS values **rti::core::RtpsReliableReaderProtocol::samples_per_app_ack** (p. 2228) and **rti::core::RtpsReliableReaderProtocol::app_ack_period** (p. 2227).

The maximum length of the response is configured using **rti::core::policy::DataReaderResourceLimits::max_app_ack_response_length** (p. 1101)

Parameters

sample_info	<<***in***>> *(p. 355)* **dds::sub::SampleInfo** (p. 2285) identifying the sample being acknowledged.
response_data	<<***in***>> *(p. 355)* Response data sent to **dds::pub::DataWriter** (p. 1138) upon acknowledgment (via **dds::pub::DataWriterListener::on_application_acknowledgment** (p. 1207))

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.53.3.41 close_contained_entities()

```
template<typename T >
void close_contained_entities ( )
```

<<***extension***>> *(p. 354)* Closes all the entities created from this **DataReader** (p. 976)

Note

Calling this function explicitly is not necessary to close a **DataReader** (p. 976).

Deletes all contained **dds::sub::cond::ReadCondition** (p. 2133), **dds::sub::cond::QueryCondition** (p. 2059), and **rti::sub::TopicQuery** (p. 2516) objects.

The operation will fail with **dds::core::PreconditionNotMetError** (p. 1941) if the any of the contained entities is in a state where it cannot be deleted.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::PreconditionNotMetError** (p. 1941)

8.53.3.42 read_noexcept()

```
template<typename T >
rti::core::Result< dds::sub::LoanedSamples< T > > read_noexcept ( ) [noexcept]
```

<<***extension***>> *(p. 354)* Alternative to **read()** (p. 996) that doesn't throw exceptions.

This operation behaves exactly like **read()** (p. 996) but is guaranteed to not throw exceptions.

As an extension function, it must be accessed with the `extensions()` method or with the `->` operator. Since `extensions()` can throw if the object is not a valid reference, the `->` operator is recommended.

Example:

```
// Call the extension function
auto result = reader->read_noexcept();
if (result.is_ok()) {
    for (const auto& sample : result.get()) {
        // ...
    }
}
```

Returns

A Result that contains the **LoanedSamples** (p. 1663) if the operation was succesful. When an error occurs, the Result contains the return code.

8.53.3.43 take_noexcept()

```
template<typename T >
rti::core::Result< dds::sub::LoanedSamples< T > > take_noexcept ( ) [noexcept]
```

<<***extension***>> *(p. 354)* Alternative to **take()** (p. 997) that doesn't throw exceptions.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This operation behaves exactly like **take()** (p. 997) but is guaranteed to not throw exceptions.

As an extension function, it must be accessed with the `extensions()` method or with the `->` operator. Since `extensions()` can throw if the object is not a valid reference, the `->` operator is recommended.

Example:

```
// Call the extension function
auto result = reader->take_noexcept();
if (result.is_ok()) {
    for (const auto& sample : result.get()) {
        // ...
    }
}
```

Returns

A Result that contains the **LoanedSamples** (p. 1663) if the operation was succesful. When an error occurs, the Result contains the return code.

8.53.3.44 read() [4/5]

```
template<typename T >
bool read (
            T & sample )
```

<<***extension***>> *(p. 354)* Copies the next not-previously-accessed valid data value from the **DataReader** (p. 976) via a read operation.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This function ignores **SampleInfo** (p. 2285) and therefore also skips samples with invalid data.

Parameters

sample	The sample where to copy the values of the next unread sample if the function returned true, or undefinded values otherwise.

Returns

True if there was an unread sample

See also

read(T& sample, dds::sub::SampleInfo& sample_info) (p. 1019)

8.53.3.45 read() [5/5]

```
template<typename T >
bool read (
            T & sample,
             dds::sub::SampleInfo & sample_info )
```

<<***extension***>> *(p. 354)* Copies the next not-previously-accessed data value from the **DataReader** (p. 976) via a read operation.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Returns

True if there was an unread sample

This operation copies the next not-previously-accessed data value from the **dds::sub::DataReader** (p. 976). This operation also copies the corresponding **dds::sub::SampleInfo** (p. 2285). The implied order among the samples stored in the **dds::sub::DataReader** (p. 976) is the same as for the **dds::sub::DataReader::read** (p. 996) operation.

The **dds::sub::DataReader::read(T& sample, dds::sub::SampleInfo& sample_info)** (p. 1019) operation is semantically equivalent to the **dds::sub::DataReader::read** (p. 996) operation, where the sample_states=NOT_READ, the view_states=ANY_VIEW_STATE, and the instance_states=ANY_INSTANCE_STATE.

Note

Calling **dds::sub::DataReader::read(T& sample, dds::sub::SampleInfo& sample_info)** (p. 1019) from the **dds::sub::DataReaderListener::on_data_available()** (p. 1058) callback reads only one sample of potentially many samples in the reader queue, because **dds::sub::DataReaderListener::on_data_available()** (p. 1058) is triggered only once when new samples arrive in the queue. Therefore, it is recommended that you call **dds::sub::DataReader::read(T& sample, dds::sub::SampleInfo& sample_info)** (p. 1019) in a loop within the on_data_available callback until **dds::sub::DataReader::read(T& sample, dds::sub::SampleInfo& sample_info)** (p. 1019) returns false . This ensures that all samples in the reader queue are serviced by application logic. (You may also choose to use **dds::sub::DataReader::read** (p. 996) rather than **dds::sub::DataReader::read(T& sample, dds::sub::SampleInfo& sample_info)** (p. 1019) in order to read more than one sample at a time.)

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::NotEnabledError** (p. 1869).

See also

dds::sub::DataReader::read (p. 996)

8.53.3.46 take() [4/5]

```
template<typename T >
bool take (
             T & sample )
```

<<***extension***>> *(p. 354)* Copies the next not-previously-accessed valid data value from the **DataReader** (p. 976) via a take operation.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This function ignores **SampleInfo** (p. 2285) and therefore also skips samples with invalid data.

Parameters

sample	The sample where to copy the values of the next unread sample if the function returned true, or undefinded values otherwise.

Returns

True if there was an unread sample to take. When this returns false, the argument is not modified.

Example:

```
Foo sample;
bool read_new_sample = reader.extensions().take(sample);
```

See also

take(T& sample, dds::sub::SampleInfo& sample_info) (p. 1021)

8.53.3.47 take() [5/5]

```
template<typename T >
bool take (
             T & sample,
              dds::sub::SampleInfo & sample_info )
```

<<***extension***>> *(p. 354)* Copies the next not-previously-accessed data value from the **DataReader** (p. 976) via a take operation.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This operation copies the next not-previously-accessed data value from the **dds::sub::DataReader** (p. 976) and 'removes' it from the **dds::sub::DataReader** (p. 976) so that it is no longer accessible. This operation also copies the corresponding **dds::sub::SampleInfo** (p. 2285). This operation is analogous to the **dds::sub::DataReader::read(↩ T& sample, dds::sub::SampleInfo& sample_info)** (p. 1019) except for the fact that the sample is removed from the **dds::sub::DataReader** (p. 976).

The **dds::sub::DataReader::take(T& sample, dds::sub::SampleInfo& sample_info)** (p. 1021) operation is semantically equivalent to the **dds::sub::DataReader::take** (p. 997) operation, where the sample_states=NOT_READ, the view_states=ANY_VIEW_STATE, and the instance_states=ANY_INSTANCE_STATE.

Note

Calling **dds::sub::DataReader::take(T& sample, dds::sub::SampleInfo& sample_info)** (p. 1021) from the **dds::sub::DataReaderListener::on_data_available()** (p. 1058) callback retrieves only one sample of potentially many samples in the reader queue, because **dds::sub::DataReaderListener::on_data_available()** (p. 1058) is triggered only once when new samples arrive in the queue. Therefore, it is recommended that you call **dds↩ ::sub::DataReader::take(T& sample, dds::sub::SampleInfo& sample_info)** (p. 1021) in a loop within the on↩ _data_available callback until **dds::sub::DataReader::take(T& sample, dds::sub::SampleInfo& sample_info)** (p. 1021) returns false . This ensures that all samples in the reader queue are serviced by application logic. (You may also choose to use the **dds::sub::DataReader::take** (p. 997) rather than **dds::sub::DataReader::take(T& sample, dds::sub::SampleInfo& sample_info)** (p. 1021) in order to take more than one sample at a time.)

Exceptions

One	of the **Standard Exceptions** (p. 428),

dds::core::NotEnabledError (p. 1869).

See also

dds::sub::DataReader::take (p. 997)

8.53.3.48 is_data_consistent() [1/2]

```
template<typename T >
bool is_data_consistent (
            const T & sample,
            const  dds::sub::SampleInfo & sample_info )
```

<<***extension***>> *(p. 354)* Checks if the sample has been overwritten by the DataWriter

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Parameters

sample	**Sample** (p. 2270) to be validated
sample_info	**dds::sub::SampleInfo** (p. 2285) object received with the sample

See also

dds::sub::DataReader::is_data_consistent (p. 1022)

8.53.3.49 is_data_consistent() [2/2]

```
template<typename T >
bool is_data_consistent (
            const  rti::sub::LoanedSample< T > & sample )
```

<<***extension***>> *(p. 354)* Checks if the sample has been overwritten by the DataWriter

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

When a sample is received via **Zero Copy transfer over shared memory** (p. 50), the sample can be reused by the DataWriter once it is removed from the DataWriter's send queue. Since there is no synchronization between the **DataReader** (p. 976) and the DataWriter, the sample could be overwritten by the DataWriter before it is processed by the **DataReader** (p. 976). The **dds::sub::DataReader::is_data_consistent** (p. 1022) operation can be used after processing the sample to check if it was overwritten by the DataWriter.

A precondition for using this operation is to set **rti::core::DataWriterShmemRefTransferModeSettings::enable_data_consistency_check** (p. 1252) to true.

Warning

This operation cannot be used when the data type is annotated with `@language_binding(FLAT_DATA)`. Reading a FlatData sample delivered with Zero Copy transfer over shared memory while the DataWriter is overwriting it is undefined behavior. An application-level synchronization mechanism is required in this case.

Parameters

sample	<<***in***>> *(p. 355)* **Sample** (p. 2270) to be validated

Returns

true if the sample is consistent (i.e., the sample has not been overwritten by the DataWriter)

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::PreconditionNotMetError** (p. 1941).

8.53.3.50 topic_name()

```
template<typename T >
const std::string & topic_name ( ) const
```

<<***extension***>> *(p. 354)* Get the topic name associated with this **DataReader** (p. 976)

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.53.3.51 type_name()

```
template<typename T >
const std::string & type_name ( ) const
```

<<***extension***>> *(p. 354)* Get the type name associated with this **DataReader** (p. 976)

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Examples

Foo.hpp.

8.53.3.52 close()

```
template<typename T >
void close ( )
```

Close this **DataReader** (p. 976).

8.53.4 Friends And Related Function Documentation

8.53.4.1 read()

```
template<typename T >
bool read (
            dds::sub::ReadModeDummyType  )  [related]
```

The stream manipulator to indicate that the reader should read samples as opposed to taking the samples.

Usage:
```
reader » read » loaned_samples;
```

See also

dds::sub::DataReader::operator >>(bool(∗manipulator)(**ReadModeDummyType** (p. 2144)))

8.53.4.2 take()

```
template<typename T >
bool take (
              dds::sub::ReadModeDummyType  )  [related]
```

The stream manipulator to indicate that the reader should take samples as opposed to reading the samples.

Usage:
```
reader » take » loaned_samples;
```

The default mode to access samples is to take, so the above is equivalent to:
```
reader » loaned_samples;
```

See also

dds::sub::DataReader::operator >>(bool(*manipulator)(**ReadModeDummyType** (p. 2144)))

8.53.4.3 max_samples()

```
template<typename T >
dds::sub::functors::MaxSamplesManipulatorFunctor max_samples (
             uint32_t n )  [related]
```

Stream manipulator to set the maximum number of samples to read or take.

Use this function to set the maximum number of samples to read/take by passing it to the **DataReader::operator >>(Functor f)** (p. 995) operator.

Parameters

n	The maximum number of samples to take

See also

dds::sub::DataReader::operator >>(Functor f) (p. 995)

8.53.4.4 content()

```
template<typename T >
dds::sub::functors::ContentFilterManipulatorFunctor content (
             const  dds::sub::Query & query )  [related]
```

Stream manipulator to set a **Query** (p. 2053) to use during the subsequent read/take operation.

The effect of using this manipulator is that the subsequent read/take will filter the samples based on the **Query** (p. 2053)'s expression. If the **DataReader** (p. 976) has no samples that meet the constraints, the read/take will not return any data.

If this stream manipulator comes before a call to the condition(const dds::sub::cond::QueryCondition& query_condition) manipulator then it will be overridden and will not have any effect on the read or take operation. Similarly, if this operation follows a call to **condition()** (p. 1026), then the previously set QueryCondition will be cleared.

Parameters

query	The **Query** (p. 2053) to use during the read/take

See also

dds::sub::DataReader::operator >>(Functor f) (p. 995)

dds::sub::condition(const dds::sub::cond::QueryCondition& query_condition)

8.53.4.5 condition()

```
template<typename T >
dds::sub::functors::ConditionManipulatorFunctor condition (
             const  dds::sub::cond::ReadCondition & condition )  [related]
```

Stream manipulator to set a QueryCondition to use during the subsequent read/take operation.

The effect of using this manipulator is that the subsequent read/take will filter the samples based on the Query↩ Conditions's expression and state. If the **DataReader** (p. 976) has no samples that meet the constraints, the read/take will not return any data.

If this stream manipulator comes before a call to the **content(const dds::sub::Query& query)** (p. 1025) manipulator then it will be overridden and will not have any effect on the read or take operation. Similarly, if this operation follows a call to **content()** (p. 1025) and/or **state(const dds::sub::status::DataState& s)** (p. 1027), then the previously set **Query** (p. 2053) and DataState will be cleared.

This manipulator is effectively a combination of the content and state manipulators.

For example:
```
reader » read
       » content(dds::sub::Query(system.reader, "foo = 7"))
       » state(dds::sub::status::DataState::new_data())
       » samples;
```

is equivalent to:
```
reader » read
       » condition(Query(system.reader, "foo = 7"), DataState()::new_data())
       » samples;
```

Parameters

condition	The QueryCondition to use during the read/take

See also

dds::sub::DataReader::operator >>(Functor f) (p. 995)

content(const dds::sub::Query& query) (p. 1025)

References **dds::sub::condition()**.

8.53.4.6 state()

```
template<typename T >
dds::sub::functors::StateFilterManipulatorFunctor state (
             const  dds::sub::status::DataState & s )  [related]
```

Stream manipulator to specify the DataState of the samples that should be read/taken.

By setting the **dds::sub::status::DataState** (p. 1118) you can specify the state of the samples that should be read or taken. The DataState of a sample encapsulates the **dds::sub::status::SampleState** (p. 2316), **dds::sub::status::ViewState** (p. 2614), and **dds::sub::status::InstanceState** (p. 1615) of a sample.

If this stream manipulator comes before a call to the condition(const dds::sub::cond::QueryCondition& query_condition) manipulator then it will be overridden and will not have any effect on the read or take operation.

Parameters

s	The DataState of the samples to be read or taken

See also

dds::sub::DataReader::operator >>(Functor f) (p. 995)

8.53.4.7 instance()

```
template<typename T >
dds::sub::functors::InstanceManipulatorFunctor instance (
            const  dds::core::InstanceHandle & h )  [related]
```

Stream manipulator to specify the instance whose samples should be read or taken.

This operation causes the subsequent read or take operation to access only samples belonging the single specified instance whose handle is `h`.

Upon successful completion, the data collection will contain samples all belonging to the same instance. The corresponding **SampleInfo** (p. 2285) verifies **SampleInfo.instance_handle()** (p. 2290) == `h`.

The subsequent read/take operation will be semantically equivalent to a read or take without specifying the instance, except in building the collection, the **DataReader** (p. 976) will check that the sample belongs to the specified instance and otherwise it will not place the sample in the returned collection.

The subsequent read/take may operation may fail with **dds::core::InvalidArgumentError** (p. 1619) if the Instance↩ Handle does not correspond to an existing data-object known to the **DataReader** (p. 976).

Parameters

h	The handle of the instance to access

See also

dds::sub::DataReader::operator >>(Functor f) (p. 995)

8.53.4.8 next_instance()

```
template<typename T >
dds::sub::functors::NextInstanceManipulatorFunctor next_instance (
            const  dds::core::InstanceHandle & h )  [related]
```

Stream manipulator to specify the samples belonging to the 'next' instance after the provided instance handle should be accessed.

This operation causes the subsequent read or take operation to access only samples belonging a single instance whose handle is considered 'next' after the provided InstanceHandle `h`.

The accessed samples will all belong to the 'next' instance with InstanceHandle 'greater' than the specified previous handle that has available samples.

This operation implies the existence of a total order 'greater-than' relationship between the instance handles. The specifics of this relationship are not all important and are implementation specific. The important thing is that, according to the middleware, all instances are ordered relative to each other. This ordering is between the instance handles; It should not depend on the state of the instance (e.g. whether it has data or not) and must be defined even for instance handles that do not correspond to instances currently managed by the **dds::sub::DataReader** (p. 976). For the purposes of the ordering, it should be 'as if' each instance handle was represented as unique integer.

The behavior of **dds::sub::DataReader::Selector::next_instance** (p. 2322) is 'as if' the **dds::sub::DataReader** (p. 976) invoked **dds::sub::instance(const dds::core::InstanceHandle& h)** (p. 665), passing the smallest instance_handle among all the ones that: (a) are greater than `previous_handle`, and (b) have available samples (i.e. samples that meet the constraints imposed by the specified states).

The special value **dds::core::InstanceHandle::nil()** (p. 1614) is guaranteed to be 'less than' any valid `instance_handle`. So the use of the parameter value `previous_handle` == **dds::core::InstanceHandle::nil()** (p. 1614) will return the samples for the instance which has the smallest `instance_handle` among all the instances that contain available samples.

Note

The operation **dds::sub::DataReader::Selector::next_instance** (p. 2322) is intended to be used in an application-driven iteration, where the application starts by passing `previous_handle` == **dds::core::InstanceHandle::nil()** (p. 1614), examines the samples returned, and then uses the `instance_handle` returned in the **dds::sub::SampleInfo** (p. 2285) as the value of the `previous_handle` argument to the next call to **dds::sub::DataReader::Selector::next_instance** (p. 2322). The iteration continues until the read/take operation doesn't return any more samples. This application-driven iteration is required to ensure that all samples on the reader queue are read.

Note that it is possible to call the **dds::sub::DataReader::Selector::next_instance** (p. 2322) operation with a `previous_handle` that does not correspond to an instance currently managed by the **dds::sub::DataReader** (p. 976). This is because as stated earlier the 'greater-than' relationship is defined even for handles not managed by the **dds::sub::DataReader** (p. 976). One practical situation where this may occur is when an application is iterating though all the instances, takes all the samples of a **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) instance, returns the loan (at which point the instance information may be removed, and thus the handle becomes invalid), and tries to read the next instance.

Parameters

h	The reference instance. The instance after this one will be selected

See also

dds::sub::DataReader::operator >>(Functor f) (p. 995)

8.53.4.9 ignore() [1/2]

```
template<typename T >
void ignore (
              dds::domain::DomainParticipant & participant,
             const  dds::core::InstanceHandle & handle )  [related]
```

Instructs RTI Connext to locally ignore a subscription.

A subscription is defined by the association of a topic name, user data, and partition set on the **dds::sub::Subscriber** (p. 2410) (see **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)). After this call, any data received related to that subscription's **dds::sub::DataReader** (p. 976) will be ignored.

This operation can be used to ignore local *and* remote DataReaders.

The subscription to ignore is identified by the `handle` argument.

- To ignore a *remote* **DataReader** (p. 976), the `handle` can be obtained from the **dds::sub::SampleInfo** (p. 2285) retrieved when reading data samples from the built-in **dds::sub::DataReader** (p. 976) for the subscription topic.

- To ignore a *local* **DataReader** (p. 976), the `handle` can be obtained by calling **dds::core::Entity::instance_handle()** (p. 1525) for the local **DataReader** (p. 976).

There is no way to reverse this operation.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::OutOfResourcesError** (p. 1898) or **dds::core::NotEnabledError** (p. 1869)

See also

dds::topic::SubscriptionBuiltinTopicData (p. 2428)

dds::topic::subscription_topic_name() (p. 444)

dds::sub::builtin_subscriber (p. 671)

Parameters

participant	The DomainParticipant for which the remote entity will be ignored
handle	The InstanceHandle of the remote entity that has to be ignored

8.53.4.10 ignore() [2/2]

```
template<typename FwdIterator >
void ignore (
              dds::domain::DomainParticipant & participant,
             FwdIterator begin,
             FwdIterator end )  [related]
```

Instructs RTI Connext to locally ignore subscriptions.

Parameters

participant	The DomainParticipant for which the remote entity will be ignored
begin	A forward iterator to the initial position in a **dds::core::InstanceHandleSeq** (p. 619) holding the handles to the remote DataReaders to be ignored.
end	A forward iterator to the final position in a **dds::core::InstanceHandleSeq** (p. 619) holding the handles to the remote DataReaders to be ignored.

See also

ignore(dds::domain::DomainParticipant&, const dds::core::InstanceHandle&) (p. 1030);

References **dds::sub::begin()**, **dds::sub::end()**, and **dds::sub::ignore()**.

8.53.4.11 matched_publications() [1/2]

```
template<typename T >
const  ::dds::core::InstanceHandleSeq matched_publications (
             const  dds::sub::DataReader< T > & reader )  [related]
```

Retrieve the list of publications currently "associated" with a **DataReader** (p. 976).

A publication is considered to be matching if all of the following criteria are true:

- The publication is within the same domain as this subscription.
- The publication has a matching **dds::topic::Topic** (p. 2474).
- The publication has compatible QoS.
- If the applications are using partitions, the publication shares a common partition with this subscription.
- The **dds::domain::DomainParticipant** (p. 1326) has not indicated that the publication's **dds::domain::DomainParticipant** (p. 1326) should be "ignored" by means of the **dds::pub::ignore** (p. 647) API.

- If the subscription is using a **dds::topic::ContentFilteredTopic** (p. 954) and the publication is using the **rti::core::policy::MultiChannel** (p. 1749), there is an intersection between at least one of the associated filters.
- If the endpoints need to exchange key material to communicate (i.e., they are securing their communications), the reader has completed the key exchange with the writer.

The handles returned in the `publication_handles` list are the ones that are used by the RTI Connext implementation to locally identify the corresponding matched **dds::pub::DataWriter** (p. 1138) entities. These handles match the ones that appear in the `instance_handle` field of the **dds::sub::SampleInfo** (p. 2285) when reading the **dds::topic::publication_topic_name()** (p. 443) builtin topic.

This API may return the publication handles of publications that are not alive. **dds::sub::DataReader::is_matched_publication_alive** (p. 1035) can be used to check the liveliness of the remote publication.

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::OutOfResourcesError** (p. 1898) if the sequence is too small and the system cannot resize it, or **dds::core::NotEnabledError** (p. 1869)

Template Parameters

T	The topic-type that the **DataReader** (p. 976) subscribes to

Parameters

reader	The reader whose publications are being retrieved

Returns

An InstanceHandleSeq containing the InstanceHandles of the matched publications for the provided **DataReader** (p. 976)

8.53.4.12 matched_publications() [2/2]

```
template<typename T , typename FwdIterator >
FwdIterator matched_publications (
             const  dds::sub::DataReader< T > & reader,
             FwdIterator begin,
             FwdIterator end )  [related]
```

Retrieve the list of publications currently "associated" with a **DataReader** (p. 976).

A publication is considered to be matching if all of the following criteria are true:

- The publication is within the same domain as this subscription.
- The publication has a matching **dds::topic::Topic** (p. 2474).
- The publication has compatible QoS.
- If the applications are using partitions, the publication shares a common partition with this subscription.
- The **dds::domain::DomainParticipant** (p. 1326) has not indicated that the publication's **dds::domain::DomainParticipant** (p. 1326) should be "ignored" by means of the **dds::pub::ignore** (p. 647) API.
- If the subscription is using a **dds::topic::ContentFilteredTopic** (p. 954) and the publication is using the **rti::core::policy::MultiChannel** (p. 1749), there is an intersection between at least one of the associated filters.
- If the endpoints need to exchange key material to communicate (i.e., they are securing their communications), the reader has completed the key exchange with the writer.

The handles returned in the `publication_handles` list are the ones that are used by the RTI Connext implementation to locally identify the corresponding matched **dds::pub::DataWriter** (p. 1138) entities. These handles match the ones that appear in the `instance_handle` field of the **dds::sub::SampleInfo** (p. 2285) when reading the **dds::topic::publication_topic_name()** (p. 443) builtin topic.

This API may return the publication handles of publications that are not alive. **dds::sub::DataReader::is_matched_publication_alive** (p. 1035) can be used to check the liveliness of the remote publication.

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::OutOfResourcesError** (p. 1898) if the sequence is too small and the system cannot resize it, or **dds::core::NotEnabledError** (p. 1869)

Template Parameters

T	The topic-type that the **DataReader** (p. 976) subscribes to
FwdIterator	A forward iterator whose value type is **dds::core::InstanceHandle** (p. 1612)

Parameters

reader	The reader whose publications are being retrieved
begin	A forward iterator to the beginning position in the destination container of matching InstanceHandles
end	A forward iterator to the ending position in the destination container of matching InstanceHandles

Returns

An iterator placed at the last position in the container where a InstanceHandle was inserted

References **dds::sub::begin()**, **dds::sub::end()**, and **dds::sub::matched_publications()**.

8.53.4.13 matched_publication_data() [1/2]

```
template<typename T >
const  dds::topic::PublicationBuiltinTopicData matched_publication_data (
             const  dds::sub::DataReader< T > & reader,
             const  dds::core::InstanceHandle & handle )  [related]
```

This operation retrieves the information on a publication that is currently "associated" with the **DataReader** (p. 976).

The publication_handle must correspond to a publication currently associated with the **dds::sub::DataReader** (p. 976). Otherwise, the operation will fail with **dds::core::InvalidArgumentError** (p. 1619). Use the operation **dds::sub::matched_publications** (p. 668) to find the publications that are currently matched with the **dds::sub::DataReader** (p. 976).

Note: This operation does not retrieve the **dds::topic::PublicationBuiltinTopicData::type** (p. 1985). This information is available through **dds::sub::DataReaderListener::on_data_available()** (p. 1058) (if a reader listener is installed on the dds::sub::DataReader<dds::topic::PublicationBuiltinTopicData>).

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869)

Template Parameters

T	The topic-type that the **DataReader** (p. 976) subscribes to

Parameters

reader	The reader associated with the publication whose data is being retrieved
handle	The InstanceHandle Handle to a specific publication associated with the DataWriter. Must correspond to a publication currently associated with the **DataReader** (p. 976).

Returns

The **dds::topic::PublicationBuiltinTopicData** (p. 1977) of the publication that is associated with the provided handle

8.53.4.14 matched_publication_participant_data()

```
template<typename T >
dds::topic::ParticipantBuiltinTopicData matched_publication_participant_data (
             const  dds::sub::DataReader< T > & reader,
             const  dds::core::InstanceHandle & handle )  [related]
```

<<***extension***>> *(p. 354)* This operation retrieves the information on the discovered **dds::domain::DomainParticipant** (p. 1326) associated with the publication that is currently matching with the **dds::sub::DataReader** (p. 976).

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

The `publication_handle` must correspond to a publication currently associated with the **dds::sub::DataReader** (p. 976). Otherwise, the operation will fail with **dds::core::InvalidArgumentError** (p. 1619). The operation may also fail with **dds::core::PreconditionNotMetError** (p. 1941) if the publication corresponds to the same **dds::domain::DomainParticipant** (p. 1326) that the **DataReader** (p. 976) belongs to. Use the operation **dds::sub::matched_publications** (p. 668) to find the publications that are currently matched with the **dds::sub::DataReader** (p. 976).

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869)

Template Parameters

T	The topic-type that the **DataReader** (p. 976) subscribes to

Parameters

reader	The reader to lookup the matched participant data of
handle	The InstanceHandle to a specific publication. Must correspond to a publication currently associated with the **DataReader** (p. 976). This handle is available in the **dds::sub::SampleInfo::publication_handle()** (p. 2290)

Returns

The **dds::topic::ParticipantBuiltinTopicData** (p. 1908) of the DomainParticipant of a matched publication of a **dds::sub::DataReader** (p. 976)

8.53.4.15 is_matched_publication_alive()

```
template<typename T >
bool is_matched_publication_alive (
            const  dds::sub::DataReader< T > & reader,
            const  dds::core::InstanceHandle & handle )  [related]
```

<<***extension***>> *(p. 354)* Check if a matched publication is alive.

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

T	The topic-type that the **DataReader** (p. 976) subscribes to.

Parameters

reader	The **DataReader** (p. 976).
handle	The **dds::core::InstanceHandle** (p. 1612) of the matched publication.

This API is used for querying the endpoint liveliness of a matched publication. A matched publication will be marked as not alive if the liveliness that it committed to through its **LIVELINESS** (p. 535) QoS policy was not respected. Note that if the participant associated with the matched publication loses liveliness, the **dds::core::InstanceHandle** (p. 1612) will become invalid and this function will fail with **dds::core::InvalidArgumentError** (p. 1619).

of the matched publication. See **dds::sub::matched_publications** (p. 668) for a description of what is considered a matched publication.

Exceptions

One	of the **Standard Exceptions** (p. 428)

Returns

A boolean indicating whether or not the matched publication is alive.

8.53.4.16 matched_publication_data() [2/2]

```
template<typename T >
std::vector<  dds::topic::PublicationBuiltinTopicData > matched_publication_data (
             const  dds::sub::DataReader< T > & reader )  [related]
```

<<***extension***>> *(p. 354)* Obtain the PublicationBuiltinTopicData for all of the publications matched with a **DataReader** (p. 976).

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

This API retrieves the matched subscription data from all of the publications currently matched a **DataReader** (p. 976).

Template Parameters

T	The topic-type that the **DataReader** (p. 976) subscribes to.

Parameters

reader	The **DataReader** (p. 976).

Returns

A std::vector containing all of the matched publication data.

8.53.4.17 find() [1/4]

```
template<typename Reader , typename FwdIterator >
uint32_t find (
             const  dds::sub::Subscriber & subscriber,
             RTI_STRING_IN topic_name,
             FwdIterator begin,
             uint32_t max_size )  [related]
```

This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching topic name.

Use this operation on the built-in **dds::sub::Subscriber** (p. 2410) (**Built-in Topics** (p. 46)) to access the built-in **dds::sub::DataReader** (p. 976) entities for the built-in topics.

The built-in **dds::sub::DataReader** (p. 976) is created when this operation is called on a built-in topic for the first time. The built-in **dds::sub::DataReader** (p. 976) is deleted automatically when the **dds::domain::DomainParticipant** (p. 1326) is deleted.

To ensure that builtin **dds::sub::DataReader** (p. 976) entities receive all the discovery traffic, it is suggested that you lookup the builtin **dds::sub::DataReader** (p. 976) before the **dds::domain::DomainParticipant** (p. 1326) is enabled. Looking up builtin **dds::sub::DataReader** (p. 976) may implicitly register builtin transports due to creation of **dds::sub::DataReader** (p. 976) (see **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered). Therefore, if you are want to modify builtin transport properties, do so *before* using this operation.

Therefore the suggested sequence when looking up builtin DataReaders is:

- Create a disabled **dds::domain::DomainParticipant** (p. 1326).
- (optional) Modify builtin transport properties
- Call **dds::sub::builtin_subscriber()** (p. 671).
- Call **dds::sub::find()** (p. 672).
- Call **enable()** (p. 1524) on the DomainParticipant.

The returned **dds::sub::DataReader** (p. 976) may be enabled or disabled.

If more than one **dds::sub::DataReader** (p. 976) is attached to the **dds::sub::Subscriber** (p. 2410), this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **dds::sub::DataReader** (p. 976) in one thread while another thread is simultaneously creating or destroying that **dds::sub::DataReader** (p. 976).

Template Parameters

Reader	The type of the reader. It can be **dds::sub::AnyDataReader** (p. 807), or an instantiation of dds::sub::DataReader<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))
FwdIterator	The type of forward iterator passed to this function

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) the **DataReader** (p. 976) belongs to
topic_name	Name of the **dds::topic::Topic** (p. 2474) associated with the **DataReader** (p. 976) that is to be looked up.
begin	A forward iterator to the position in the destination container where the DataReaders will be copied into
max_size	Only 1 **DataReader** (p. 976) will be returned from this function

Returns

The number of DataReaders that were found (either 0 or 1)

References **dds::sub::begin()**.

8.53.4.18 find() [2/4]

```
template<typename Reader , typename BinIterator >
uint32_t find (
             const  dds::sub::Subscriber & subscriber,
             RTI_STRING_IN topic_name,
             BinIterator begin )  [related]
```

This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching topic name.

Use this operation on the built-in **dds::sub::Subscriber** (p. 2410) (**Built-in Topics** (p. 46)) to access the built-in **dds::sub::DataReader** (p. 976) entities for the built-in topics.

The built-in **dds::sub::DataReader** (p. 976) is created when this operation is called on a built-in topic for the first time. The built-in **dds::sub::DataReader** (p. 976) is deleted automatically when the **dds::domain::DomainParticipant** (p. 1326) is deleted.

To ensure that builtin **dds::sub::DataReader** (p. 976) entities receive all the discovery traffic, it is suggested that you lookup the builtin **dds::sub::DataReader** (p. 976) before the **dds::domain::DomainParticipant** (p. 1326) is enabled. Looking up builtin **dds::sub::DataReader** (p. 976) may implicitly register builtin transports due to creation of **dds::sub::DataReader** (p. 976) (see **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered). Therefore, if you are want to modify builtin transport properties, do so *before* using this operation.

Therefore the suggested sequence when looking up builtin DataReaders is:

- Create a disabled **dds::domain::DomainParticipant** (p. 1326).
- (optional) Modify builtin transport properties
- Call **dds::sub::builtin_subscriber()** (p. 671).
- Call **dds::sub::find()** (p. 672).
- Call **enable()** (p. 1524) on the DomainParticipant.

The returned **dds::sub::DataReader** (p. 976) may be enabled or disabled.

If more than one **dds::sub::DataReader** (p. 976) is attached to the **dds::sub::Subscriber** (p. 2410), this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **dds::sub::DataReader** (p. 976) in one thread while another thread is simultaneously creating or destroying that **dds::sub::DataReader** (p. 976).

Template Parameters

Reader	The type of the reader. It can be **dds::sub::AnyDataReader** (p. 807), or an instantiation of dds::sub::DataReader<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))
BinIterator	The type of back-inserting iterator passed to this function

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) the **DataReader** (p. 976) belongs to
topic_name	Name of the **dds::topic::Topic** (p. 2474) associated with the **DataReader** (p. 976) that is to be looked up.
begin	A back-inserting iterator to the position in the destination container to insert the found **DataReader** (p. 976) into

Returns

The number of DataReaders that were found (either 0 or 1)

References **dds::sub::begin()**.

8.53.4.19 find() [3/4]

```
template<typename READER , typename T , typename FwdIterator >
uint32_t find (
            const  dds::sub::Subscriber & subscriber,
```

```
            const  dds::topic::TopicDescription< T > & topic_description,
            FwdIterator begin,
            uint32_t max_size )  [related]
```

This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching TopicDescription.

See also

find(const dds::sub::Subscriber& subscriber,RTI_STRING_IN topic_name,FwdIterator begin,uint32_t max_size) (p. 1037)

References **dds::sub::begin()**, and **dds::sub::find()**.

8.53.4.20 find() [4/4]

```
template<typename READER , typename T , typename BinIterator >
uint32_t find (
            const  dds::sub::Subscriber & subscriber,
            const  dds::topic::TopicDescription< T > & topic_description,
            BinIterator begin )  [related]
```

This function retrieves a previously-created **DataReader** (p. 976) belonging to the **Subscriber** (p. 2410) that is attached to a Topic with a matching TopicDescription.

See also

uint32_t **find(const dds::sub::Subscriber& subscriber,RTI_STRING_IN topic_name,BinIterator begin)** (p. 1038)

References **dds::sub::begin()**, and **dds::sub::find()**.

8.53.4.21 find_datareaders() [1/2]

```
template<typename AnyDataReaderBackInsertIterator >
uint32_t find_datareaders (
             dds::sub::Subscriber subscriber,
            AnyDataReaderBackInsertIterator begin )  [related]
```

<<***extension***>> *(p. 354)* Retrieve all the **dds::sub::DataReader** (p. 976) created from this **dds::sub::Subscriber** (p. 2410)

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

AnyDataReaderBackInsertIterator	Type of the back-inserting iterator passed into this function, whose value_type must be **dds::sub::AnyDataReader** (p. 807).

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) the DataReaders belong to
begin	A back-inserting iterator to the position in the destination container to insert the found DataReaders into

Returns

The number of found DataReaders

See also

Looking up DataReaders (p. 322)

References **rti::sub::begin()**.

8.53.4.22 find_datareaders() [2/2]

```
template<typename AnyDataReaderForwardIterator >
uint32_t find_datareaders (
             dds::sub::Subscriber subscriber,
            AnyDataReaderForwardIterator begin,
            uint32_t max_size )  [related]
```

<<***extension***>> *(p. 354)* Retrieve all the readers created from a subscriber.

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

AnyDataReaderForwardIterator	A forward iterator whose `value_type` is **dds::sub::AnyDataReader** (p. 807)

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) the readers belong to
begin	A forward iterator to the position in the destination container where to begin copying the found readers into.

Parameters

max_size	The maximum number of readers to return

Returns

The number of found readers

See also

Looking up DataReaders (p. 322)

References **rti::sub::begin()**.

8.53.4.23 find_datareader_by_topic_name()

```
template<typename Reader >
Reader find_datareader_by_topic_name (
              dds::sub::Subscriber subscriber,
             const std::string & topic_name )  [related]
```

<<***extension***>> *(*p. *354)* Retrieves a **dds::sub::DataReader** (p. 976) with the given topic name within the **dds::sub::Subscriber** (p. 2410)

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Use this operation on the built-in **dds::sub::Subscriber** (p. 2410) (**Built-in Topics** (p. 46)) to access the built-in **dds::sub::DataReader** (p. 976) entities for the built-in topics.

The built-in **dds::sub::DataReader** (p. 976) is created when this operation is called on a built-in topic for the first time. The built-in **dds::sub::DataReader** (p. 976) is deleted automatically when the **dds::domain::DomainParticipant** (p. 1326) is deleted.

To ensure that builtin **dds::sub::DataReader** (p. 976) entities receive all the discovery traffic, it is suggested that you lookup the builtin **dds::sub::DataReader** (p. 976) before the **dds::domain::DomainParticipant** (p. 1326) is enabled. Looking up builtin **dds::sub::DataReader** (p. 976) may implicitly register builtin transports due to creation of **dds::sub::DataReader** (p. 976) (see **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered). Therefore, if you are want to modify builtin transport properties, do so *before* using this operation.

Therefore the suggested sequence when looking up builtin DataReaders is:

- Create a disabled **dds::domain::DomainParticipant** (p. 1326).
- (optional) Modify builtin transport properties
- Call **dds::sub::builtin_subscriber()** (p. 671).
- Call **dds::sub::find()** (p. 672).
- Call **enable()** (p. 1524) on the DomainParticipant.

The returned **dds::sub::DataReader** (p. 976) may be enabled or disabled.

If more than one **dds::sub::DataReader** (p. 976) is attached to the **dds::sub::Subscriber** (p. 2410), this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **dds::sub::DataReader** (p. 976) in one thread while another thread is simultaneously creating or destroying that **dds::sub::DataReader** (p. 976).

Template Parameters

Reader	The type of the reader. It can be **dds::sub::AnyDataReader** (p. 807), or an instantiation of dds::sub::DataReader<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) that created the **DataReader** (p. 976) to find
topic_name	Topic name of the **DataReader** (p. 976) to find

Returns

The **DataReader** (p. 976) with the given topic name, or **dds::core::null** (p. 438) if it doesn't exist

See also

Looking up DataReaders (p. 322)

References **dds::core::null**.

8.53.4.24 find_datareader_by_name() [1/2]

```
template<typename Reader >
Reader find_datareader_by_name (
             dds::sub::Subscriber subscriber,
            const std::string & datareader_name )  [related]
```

<<***extension***>> *(p. 354)* Retrieves a **dds::sub::DataReader** (p. 976) with the given name within the **dds::sub::Subscriber** (p. 2410)

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

Reader	The type of the reader. It can be **dds::sub::AnyDataReader** (p. 807), or an instantiation of dds::sub::DataReader<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))

Every **dds::sub::DataReader** (p. 976) in the system has an entity name which is configured and stored in the <<***extension***>> *(p. 354)* EntityName policy, **ENTITY_NAME** (p. 532).

This operation retrieves the **dds::sub::DataReader** (p. 976) within the **dds::sub::Subscriber** (p. 2410) whose name matches the one specified. If there are several **dds::sub::DataReader** (p. 976) with the same name within the **dds::sub::Subscriber** (p. 2410), the operation returns the first matching occurrence.

Parameters

subscriber	The **dds::sub::Subscriber** (p. 2410) that created the **DataReader** (p. 976) to find
datareader_name	Entity name of the **DataReader** (p. 976) to find

Returns

The **DataReader** (p. 976) with the given name, or **dds::core::null** (p. 438) if it doesn't exist

See also

Looking up DataReaders (p. 322)

References **dds::core::null**.

8.53.4.25 find_datareader_by_topic_description()

```
template<typename Reader , typename T >
Reader find_datareader_by_topic_description (
            const  dds::sub::Subscriber & subscriber,
            const  dds::topic::TopicDescription< T > & topic_description )  [related]
```

<<***extension***>> *(p. 354)* Retrieves a **dds::sub::DataReader** (p. 976) with the given TopicDescription within the **dds::sub::Subscriber** (p. 2410)

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

Reader	The type of the reader returned, for example, dds::sub::DataReader<Foo>, or **dds::sub::AnyDataReader** (p. 807)
T	The topic-type

Parameters

Parameters

subscriber	The **Subscriber** (p. 2410) to which the **DataReader** (p. 976) belongs
topic_description	The TopicDescription identifying the **DataReader** (p. 976) to find

Returns

The found **DataReader** (p. 976), or **dds::core::null** (p. 438) if it doesn't exist

References **dds::core::null**.

8.53.4.26 find_datareader_by_name() [2/2]

```
template<typename Reader >
Reader find_datareader_by_name (
              dds::domain::DomainParticipant participant,
             const std::string & datareader_name )  [related]
```

<<***extension***>> *(p. 354)* Retrieves a **dds::sub::DataReader** (p. 976) within the **dds::domain::DomainParticipant** (p. 1326) with the given name

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

Reader	The type of the reader. It can be **dds::sub::AnyDataReader** (p. 807), or an instantiation of dds::sub::DataReader<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))

Every **dds::sub::DataReader** (p. 976) in the system has an entity name which is configured and stored in the Entity↩ Name policy, **ENTITY_NAME** (p. 532).

Every **dds::sub::Subscriber** (p. 2410) in the system has an entity name which is also configured and stored in the EntityName policy, **ENTITY_NAME** (p. 532).

This operation retrieves a **dds::sub::DataReader** (p. 976) within a **dds::sub::Subscriber** (p. 2410) given the specified name which encodes both to the **dds::sub::DataReader** (p. 976) and the **dds::sub::Subscriber** (p. 2410) name.

If there are several **dds::sub::DataReader** (p. 976) with the same name within the corresponding **dds::sub::Subscriber** (p. 2410) this function returns the first matching occurrence.

The specified name might be given as a fully-qualified entity name or as a plain name.

The fully qualified entity name is a concatenation of the **dds::sub::Subscriber** (p. 2410) to which the **dds::sub::Data↩ Reader** (p. 976) belongs and the entity name of of the **dds::sub::DataReader** (p. 976) itself, separated by a double colon "::". For example: MySubscriberName::MyDataReaderName

The plain name contains the **dds::sub::DataReader** (p. 976) name only. In this situation it is implied that the **dds::sub↩ ::DataReader** (p. 976) belongs to the implicit **dds::sub::Subscriber** (p. 2410) so the use of a plain name is equivalent to specifying a fully qualified name with the **dds::sub::Subscriber** (p. 2410) name part being "implicit". For example: the plain name "MyDataReaderName" is equivalent to specifiying the fully qualified name "implicit::MyDataReaderName"

The **dds::sub::DataReader** (p. 976) is only looked up within the **dds::sub::Subscriber** (p. 2410) specified in the fully qualified name, or within the implicit **dds::sub::Subscriber** (p. 2410) if the name was not fully qualified.

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) within which the **dds::sub::DataReader** (p. 976) exists
datareader_name	EntityName of the **DataReader** (p. 976) to find

Returns

The first reader with the given name or **dds::core::null** (p. 438) if it is not found.

See also

rti::sub::find_datareader_by_name(const dds::sub::Subscriber&, const std::string&)

References **dds::core::null**.

8.54 rti::core::status::DataReaderCacheStatus Class Reference

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::datareader_cache()** (p. 2386)

```
#include <Status.hpp>
```

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- int64_t **sample_count** () const

  The number of samples in the reader's queue.
- int64_t **sample_count_peak** () const

  The highest number of samples in the reader's queue over the lifetime of the reader.
- int64_t **old_source_timestamp_dropped_sample_count** () const

*The number of samples dropped as a result of receiving a sample older than the last one, using **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264).*

- int64_t **tolerance_source_timestamp_dropped_sample_count** () const

  *The number of samples dropped as a result of receiving a sample in the future, using **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264).*

- int64_t **ownership_dropped_sample_count** () const

  The number of samples dropped as a result of receiving a sample from a DataWriter with a lower strength, using Exclusive Ownership.

- int64_t **content_filter_dropped_sample_count** () const

  The number of user samples filtered by the DataReader due to Content-Filtered Topics.

- int64_t **time_based_filter_dropped_sample_count** () const

  *The number of user samples filtered by the DataReader due to **dds::core::policy::TimeBasedFilter** (p. 2469).*

- int64_t **expired_dropped_sample_count** () const

  *The number of samples expired by the DataReader due to **dds::core::policy::Lifespan** (p. 1635) or the autopurge sample delays.*

- int64_t **virtual_duplicate_dropped_sample_count** () const

  The number of virtual duplicate samples dropped by the DataReader. A sample is a virtual duplicate if it has the same identity (Virtual Writer GUID and Virtual Sequence Number) as a previously received sample.

- int64_t **replaced_dropped_sample_count** () const

  *The number of samples replaced by the DataReader due to **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607) replacement.*

- int64_t **writer_removed_batch_sample_dropped_sample_count** () const

  The number of batch samples received by the DataReader that were marked as removed by the DataWriter.

- int64_t **total_samples_dropped_by_instance_replacement** () const

  *The number of samples with sample state **dds::sub::status::SampleState::not_read()** (p. 2318) that were dropped when removing an instance due to instance replacement. See **rti::core::policy::DataReaderResourceLimits::instance_replacement** (p. 1106) for more details about when instances are replaced.*

- int64_t **alive_instance_count** () const

  *The number of instances in the DataReader's queue with an instance state equal to **dds::sub::status::InstanceState::alive()** (p. 1617).*

- int64_t **alive_instance_count_peak** () const

  *The highest value of **rti::core::status::DataReaderCacheStatus::alive_instance_count** (p. 1051) over the lifetime of the DataReader.*

- int64_t **no_writers_instance_count** () const

  *The number of instances in the DataReader's queue with an instance state equal to **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617).*

- int64_t **no_writers_instance_count_peak** () const

  *The highest value of **rti::core::status::DataReaderCacheStatus::no_writers_instance_count** (p. 1051) over the lifetime of the DataReader.*

- int64_t **disposed_instance_count** () const

  *The number of instances in the DataReader's queue with an instance state equal to **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617).*

- int64_t **disposed_instance_count_peak** () const

  *The highest value of **rti::core::status::DataReaderCacheStatus::disposed_instance_count** (p. 1052) over the lifetime of the DataReader.*

- int64_t **detached_instance_count** () const

  The number of minimal instance states currently being maintained in the DataReader's queue.

- int64_t **detached_instance_count_peak** () const

  *The highest value of **rti::core::status::DataReaderCacheStatus::detached_instance_count** (p. 1052) over the lifetime of the DataReader.*

- int64_t **compressed_sample_count** () const

  The number of received samples compressed by a DataWriter.

8.54.1 Detailed Description

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::datareader_cache()** (p. 2386)

Entity:

dds::sub::DataReader (p. 976)

8.54.2 Member Function Documentation

8.54.2.1 sample_count()

```
int64_t rti::core::status::DataReaderCacheStatus::sample_count ( ) const  [inline]
```

The number of samples in the reader's queue.

includes samples that may not yet be available to be read or taken by the user, due to samples being received out of order or **PRESENTATION** (p. 540)

8.54.2.2 sample_count_peak()

```
int64_t rti::core::status::DataReaderCacheStatus::sample_count_peak ( ) const  [inline]
```

The highest number of samples in the reader's queue over the lifetime of the reader.

8.54.2.3 old_source_timestamp_dropped_sample_count()

```
int64_t rti::core::status::DataReaderCacheStatus::old_source_timestamp_dropped_sample_count ( )
const  [inline]
```

The number of samples dropped as a result of receiving a sample older than the last one, using **dds::core::policy::↩ DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264).

When the DataReader is using **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264):

- If the DataReader receives a sample for an instance with a source timestamp that is older than the last source timestamp received for the instance, the sample is dropped.
- If the DataReader receives a sample for an instance with a source timestamp that is equal to the last source timestamp received for the instance and the writer has a higher virtual GUID, the sample is dropped.

8.54.2.4 tolerance_source_timestamp_dropped_sample_count()

```
int64_t rti::core::status::DataReaderCacheStatus::tolerance_source_timestamp_dropped_sample_count
( ) const  [inline]
```

The number of samples dropped as a result of receiving a sample in the future, using **dds::core::policy::Destination↩ OrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264).

When the DataReader is using **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264): the DataReader will accept a sample only if the source timestamp is no farther in the future from the reception timestamp than the source_timestamp_tolerance. Otherwise, the sample is dropped.

8.54.2.5 ownership_dropped_sample_count()

```
int64_t rti::core::status::DataReaderCacheStatus::ownership_dropped_sample_count ( ) const  [inline]
```

The number of samples dropped as a result of receiving a sample from a DataWriter with a lower strength, using Exclusive Ownership.

When using Exclusive Ownership, the DataReader receives data from multiple DataWriters. Each instance can only be owned by one DataWriter.

If other DataWriters write samples on this instance, the samples will be dropped.

8.54.2.6 content_filter_dropped_sample_count()

```
int64_t rti::core::status::DataReaderCacheStatus::content_filter_dropped_sample_count ( ) const
[inline]
```

The number of user samples filtered by the DataReader due to Content-Filtered Topics.

When using a content filter on the DataReader side, if the sample received by the DataReader does not pass the filter, it will be dropped.

8.54.2.7 time_based_filter_dropped_sample_count()

```
int64_t rti::core::status::DataReaderCacheStatus::time_based_filter_dropped_sample_count ( ) const
[inline]
```

The number of user samples filtered by the DataReader due to **dds::core::policy::TimeBasedFilter** (p. 2469).

When using **TIME_BASED_FILTER** (p. 547) on the DataReader side, if the sample received by the DataReader does not pass the minimum_separation filter, it will be dropped.

8.54.2.8 expired_dropped_sample_count()

```
int64_t rti::core::status::DataReaderCacheStatus::expired_dropped_sample_count ( ) const  [inline]
```

The number of samples expired by the DataReader due to **dds::core::policy::Lifespan** (p. 1635) or the autopurge sample delays.

- **dds::core::policy::Lifespan** (p. 1635)
  When a sample expires due to **dds::core::policy::Lifespan** (p. 1635), the data is removed from the DataReader caches. This sample will be considered dropped if its **dds::sub::status::SampleState** (p. 2316) was **dds::sub↩ ::status::SampleState::not_read()** (p. 2318).
- **dds::core::policy::ReaderDataLifecycle::autopurge_nowriter_samples_delay** (p. 2140)
  When a sample expires due to **dds::core::policy::ReaderDataLifecycle::autopurge_nowriter_samples_delay** (p. 2140), this sample will be considered dropped if its **dds::sub::status::SampleState** (p. 2316) was **dds::sub↩ ::status::SampleState::not_read()** (p. 2318).
- **dds::core::policy::ReaderDataLifecycle::autopurge_disposed_samples_delay** (p. 2141)
  When a sample expires due to **dds::core::policy::ReaderDataLifecycle::autopurge_disposed_samples_↩ delay** (p. 2141), this sample will be considered dropped if its **dds::sub::status::SampleState** (p. 2316) was **dds↩ ::sub::status::SampleState::not_read()** (p. 2318).

8.54.2.9 virtual_duplicate_dropped_sample_count()

```
int64_t rti::core::status::DataReaderCacheStatus::virtual_duplicate_dropped_sample_count ( ) const
[inline]
```

The number of virtual duplicate samples dropped by the DataReader. A sample is a virtual duplicate if it has the same identity (Virtual Writer GUID and Virtual Sequence Number) as a previously received sample.

When two DataWriters with the same logical data source publish a sample with the same sequence_number, one sample will be dropped and the other will be received by the DataReader.

This can happen when multiple writers are writing on behalf of the same original DataWriter: for example, in systems with redundant Routing Services or when a DataReader is receiving samples both directly from the original DataWriter and from an instance of Persistence Service.

8.54.2.10 replaced_dropped_sample_count()

```
int64_t rti::core::status::DataReaderCacheStatus::replaced_dropped_sample_count ( ) const  [inline]
```

The number of samples replaced by the DataReader due to **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607) replacement.

When the number of samples for an instance in the queue reaches the **dds::core::policy::History::depth** (p. 1605) value, a new sample for the instance will replace the oldest sample for the instance in the queue.

The new sample will be accepted and the old sample will be dropped.

This counter will only be updated if the replaced sample's **dds::sub::status::SampleState** (p. 2316) was **dds::sub↩ ::status::SampleState::not_read()** (p. 2318).

8.54.2.11 writer_removed_batch_sample_dropped_sample_count()

```
int64_t rti::core::status::DataReaderCacheStatus::writer_removed_batch_sample_dropped_sample_count
( ) const  [inline]
```

The number of batch samples received by the DataReader that were marked as removed by the DataWriter.

When the DataReader receives a batch, the batch can contain samples marked as removed by the DataWriter. Examples of removed samples in a batch could be because of sample replacement due to **dds::core::policy::History↩ Kind_def::KEEP_LAST** (p. 1607) **dds::core::policy::History** (p. 1602) QoS on the DataWriter or because the duration in **dds::core::policy::Lifespan** (p. 1635) was reached. By default, any sample marked as removed from a batch is dropped (unless you set the dds.data_reader.accept_writer_removed_batch_samples property in the **rti::core↩ ::policy::Property** (p. 1969) to true). Note: Historical data with removed batch samples written before the DataReader joined the DDS domain will also be included in the count.

8.54.2.12 total_samples_dropped_by_instance_replacement()

```
int64_t rti::core::status::DataReaderCacheStatus::total_samples_dropped_by_instance_replacement (
) const  [inline]
```

The number of samples with sample state **dds::sub::status::SampleState::not_read()** (p. 2318) that were dropped when removing an instance due to instance replacement. See **rti::core::policy::DataReaderResourceLimits↩ ::instance_replacement** (p. 1106) for more details about when instances are replaced.

8.54.2.13 alive_instance_count()

```
int64_t rti::core::status::DataReaderCacheStatus::alive_instance_count ( ) const  [inline]
```

The number of instances in the DataReader's queue with an instance state equal to **dds::sub::status::InstanceState↩ ::alive()** (p. 1617).

8.54.2.14 alive_instance_count_peak()

```
int64_t rti::core::status::DataReaderCacheStatus::alive_instance_count_peak ( ) const  [inline]
```

The highest value of **rti::core::status::DataReaderCacheStatus::alive_instance_count** (p. 1051) over the lifetime of the DataReader.

See also

rti::core::status::DataReaderCacheStatus::alive_instance_count (p. 1051)

8.54.2.15 no_writers_instance_count()

```
int64_t rti::core::status::DataReaderCacheStatus::no_writers_instance_count ( ) const  [inline]
```

The number of instances in the DataReader's queue with an instance state equal to **dds::sub::status::InstanceState↩ ::not_alive_no_writers()** (p. 1617).

8.54.2.16 no_writers_instance_count_peak()

```
int64_t rti::core::status::DataReaderCacheStatus::no_writers_instance_count_peak ( ) const  [inline]
```

The highest value of **rti::core::status::DataReaderCacheStatus::no_writers_instance_count** (p. 1051) over the lifetime of the DataReader.

See also

rti::core::status::DataReaderCacheStatus::no_writers_instance_count (p. 1051)

8.54.2.17 disposed_instance_count()

```
int64_t rti::core::status::DataReaderCacheStatus::disposed_instance_count ( ) const  [inline]
```

The number of instances in the DataReader's queue with an instance state equal to **dds::sub::status::InstanceState↩ ::not_alive_disposed()** (p. 1617).

8.54.2.18 disposed_instance_count_peak()

```
int64_t rti::core::status::DataReaderCacheStatus::disposed_instance_count_peak ( ) const  [inline]
```

The highest value of **rti::core::status::DataReaderCacheStatus::disposed_instance_count** (p. 1052) over the lifetime of the DataReader.

See also

rti::core::status::DataReaderCacheStatus::disposed_instance_count (p. 1052)

8.54.2.19 detached_instance_count()

```
int64_t rti::core::status::DataReaderCacheStatus::detached_instance_count ( ) const  [inline]
```

The number of minimal instance states currently being maintained in the DataReader's queue.

If **rti::core::policy::DataReaderResourceLimits::keep_minimum_state_for_instances** (p. 1101) is true, the DataReader will keep up to a maximum of **rti::core::policy::DataReaderResourceLimits::max_total_instances** (p. 1096) detached instances in its queue. For a more in-depth description of detached instances, refer to **rti::core::policy::DataReaderResourceLimits::max_total_instances** (p. 1096).

8.54.2.20 detached_instance_count_peak()

```
int64_t rti::core::status::DataReaderCacheStatus::detached_instance_count_peak ( ) const  [inline]
```

The highest value of **rti::core::status::DataReaderCacheStatus::detached_instance_count** (p. 1052) over the lifetime of the DataReader.

See also

rti::core::status::DataReaderCacheStatus::detached_instance_count (p. 1052)

8.54.2.21 compressed_sample_count()

```
int64_t rti::core::status::DataReaderCacheStatus::compressed_sample_count ( ) const  [inline]
```

The number of received samples compressed by a DataWriter.

These include data, dispose, and unregister samples sent by a DataWriter.

8.55 rti::core::policy::DataReaderInstanceRemovalKind_def Struct Reference

Sets the kinds of instances that can be replaced when instance resource limits (**dds::core::policy::ResourceLimits::max_instances** (p. 2206)) are reached.

```
#include <PolicyKind.hpp>
```

Public Types

- enum **type** {
  NO_INSTANCE ,
  EMPTY_INSTANCES ,
  FULLY_PROCESSED_INSTANCES ,
  ANY_INSTANCE }

  The underlying `enum` *type.*

8.55.1 Detailed Description

Sets the kinds of instances that can be replaced when instance resource limits (**dds::core::policy::ResourceLimits↩ ::max_instances** (p. 2206)) are reached.

See also

rti::core::policy::DataReaderResourceLimits::instance_replacement (p. 1106)

8.55.2 Member Enumeration Documentation

8.55.2.1 type

`enum` **rti::core::policy::DataReaderInstanceRemovalKind_def::type**

The underlying `enum` type.

Enumerator

NO_INSTANCE	No instance can be removed. If an instance resource is required because **dds::core::policy::ResourceLimits::max_instances** (p. 2206) is reached, this setting will disallow instances from being replaced. Samples for new instances will be dropped and reported as lost with reason **rti::core::status::SampleLostState::lost_by_instances_limit()** (p. 2299).
EMPTY_INSTANCES	Only empty instances can be removed. Instances can be replaced only if they are empty. An instance is considered empty when all samples have been taken or removed from the DataReader queue due to the **dds::core::policy::Lifespan** (p. 1635) or sample purging due to the **dds::core::policy::ReaderDataLifecycle** (p. 2137), and there are no outstanding loans on any of the instance's samples.

Enumerator

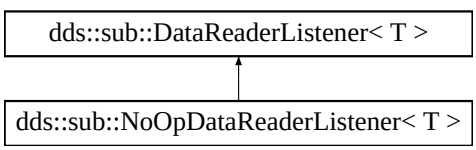
FULLY_PROCESSED_INSTANCES	Only fully-processed instances can be removed. An instance is considered fully processed if every sample for the instance has been processed by the application. A sample is considered processed by the application depending on the **dds::core::policy::Reliability::kind** (p. 2153): • **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) (depends on the **rti::core::policy::AcknowledgmentKind_def** (p. 797)): – **rti::core::policy::AcknowledgmentKind_def::PROTOCOL** (p. 797) or **rti::core::policy::AcknowledgmentKind_def::↩ APPLICATION_AUTO** (p. 798): The sample is considered processed when it has been read or taken by the application and return_loan has been called. – **rti::core::policy::AcknowledgmentKind_def::↩ APPLICATION_EXPLICIT** (p. 798): The sample is considered processed when the subscribing application has explicitly acknowledged the DDS sample, the AppAckConf has been received, and the application has called return_loan. • **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157): All samples are considered processed when they have been read or taken by the application and return_loan has been called.
ANY_INSTANCE	Any instance can be removed. Instances can be replaced regardless of whether the subscribing application has processed all of the samples. Samples that have not been processed will be removed.

8.56 dds::sub::DataReaderListener< T > Class Template Reference

The **Listener** (p. 1637) to notify status changes for a **dds::sub::DataReader** (p. 976).

```
#include <dds/sub/DataReaderListener.hpp>
```

Inheritance diagram for dds::sub::DataReaderListener< T >:



Public Member Functions

- virtual void **on_requested_deadline_missed** (**DataReader**< T > &reader, const **dds::core::status::↩ RequestedDeadlineMissedStatus** &status)=0

*Handles the **dds::core::status::RequestedDeadlineMissedStatus** (p. 2184) status.*

- virtual void **on_requested_incompatible_qos** (**DataReader**< T > &reader, const **dds::core::status::RequestedIncompatibleQosStatus** &status)=0

  *Handles the **dds::core::status::RequestedIncompatibleQosStatus** (p. 2185) status.*
- virtual void **on_sample_rejected** (**DataReader**< T > &reader, const **dds::core::status::SampleRejectedStatus** &status)=0

  *Handles the **dds::core::status::SampleRejectedStatus** (p. 2315) status.*
- virtual void **on_liveliness_changed** (**DataReader**< T > &reader, const **dds::core::status::LivelinessChangedStatus** &status)=0

  *Handles the **dds::core::status::LivelinessChangedStatus** (p. 1652) status.*
- virtual void **on_data_available** (**DataReader**< T > &reader)=0

  Called when one or more new data samples have been received.
- virtual void **on_subscription_matched** (**DataReader**< T > &reader, const **dds::core::status::SubscriptionMatchedStatus** &status)=0

  *Handles the **dds::core::status::SubscriptionMatchedStatus** (p. 2439) status.*
- virtual void **on_sample_lost** (**DataReader**< T > &reader, const **dds::core::status::SampleLostStatus** &status)=0

  *Handles the **dds::core::status::SampleLostStatus** (p. 2303) status.*

8.56.1 Detailed Description

template<typename T>
class dds::sub::DataReaderListener< T >

The **Listener** (p. 1637) to notify status changes for a **dds::sub::DataReader** (p. 976).

Entity:

dds::sub::DataReader (p. 976)

Status:

dds::core::status::StatusMask::data_available() (p. 2383);
dds::core::status::StatusMask::liveliness_changed() (p. 2383), **dds::core::status::LivelinessChangedStatus** (p. 1652);
dds::core::status::StatusMask::requested_deadline_missed() (p. 2380), **dds::core::status::RequestedDeadlineMissedStatus** (p. 2184);
dds::core::status::StatusMask::requested_incompatible_qos() (p. 2381), **dds::core::status::RequestedIncompatibleQosStatus** (p. 2185);
dds::core::status::StatusMask::sample_lost() (p. 2381), **dds::core::status::SampleLostStatus** (p. 2303);
dds::core::status::StatusMask::sample_rejected() (p. 2382), **dds::core::status::SampleRejectedStatus** (p. 2315);
dds::core::status::StatusMask::subscription_matched() (p. 2384), **dds::core::status::SubscriptionMatchedStatus** (p. 2439);

See also

Status Kinds (p. 429)

Operations Allowed in Listener Callbacks (p. 1639)

See also

NoOpDataReaderListener (p. 1838)

8.56.2 Member Function Documentation

8.56.2.1 on_requested_deadline_missed()

```
template<typename T >
virtual void  dds::sub::DataReaderListener< T >::on_requested_deadline_missed (
              DataReader< T > & reader,
             const  dds::core::status::RequestedDeadlineMissedStatus & status )  [pure virtual]
```

Handles the **dds::core::status::RequestedDeadlineMissedStatus** (p. 2184) status.

Implemented in **dds::sub::NoOpDataReaderListener< RequestType >** (p. 1839), and **dds::sub::NoOpDataReaderListener< T >** (p. 1839).

8.56.2.2 on_requested_incompatible_qos()

```
template<typename T >
virtual void  dds::sub::DataReaderListener< T >::on_requested_incompatible_qos (
              DataReader< T > & reader,
             const  dds::core::status::RequestedIncompatibleQosStatus & status )  [pure virtual]
```

Handles the **dds::core::status::RequestedIncompatibleQosStatus** (p. 2185) status.

Implemented in **dds::sub::NoOpDataReaderListener< RequestType >** (p. 1839), and **dds::sub::NoOpDataReaderListener< T >** (p. 1839).

8.56.2.3 on_sample_rejected()

```
template<typename T >
virtual void  dds::sub::DataReaderListener< T >::on_sample_rejected (
              DataReader< T > & reader,
             const  dds::core::status::SampleRejectedStatus & status )  [pure virtual]
```

Handles the **dds::core::status::SampleRejectedStatus** (p. 2315) status.

Implemented in **dds::sub::NoOpDataReaderListener< RequestType >** (p. 1839), and **dds::sub::NoOpDataReaderListener< T >** (p. 1839).

8.56.2.4 on_liveliness_changed()

```
template<typename T >
virtual void  dds::sub::DataReaderListener< T >::on_liveliness_changed (
              DataReader< T > & reader,
             const  dds::core::status::LivelinessChangedStatus & status )  [pure virtual]
```

Handles the **dds::core::status::LivelinessChangedStatus** (p. 1652) status.

Implemented in **dds::sub::NoOpDataReaderListener< RequestType >** (p. 1839), and **dds::sub::NoOpDataReaderListener< T >** (p. 1839).

8.56.2.5 on_data_available()

```
template<typename T >
virtual void  dds::sub::DataReaderListener< T >::on_data_available (
              DataReader< T > & reader )  [pure virtual]
```

Called when one or more new data samples have been received.

Implemented in **dds::sub::NoOpDataReaderListener< RequestType >** (p. 1840), and **dds::sub::NoOpDataReaderListener< T >** (p. 1840).

8.56.2.6 on_subscription_matched()

```
template<typename T >
virtual void  dds::sub::DataReaderListener< T >::on_subscription_matched (
              DataReader< T > & reader,
             const  dds::core::status::SubscriptionMatchedStatus & status )  [pure virtual]
```

Handles the **dds::core::status::SubscriptionMatchedStatus** (p. 2439) status.

Implemented in **dds::sub::NoOpDataReaderListener< RequestType >** (p. 1840), and **dds::sub::NoOpDataReaderListener< T >** (p. 1840).

8.56.2.7 on_sample_lost()

```
template<typename T >
virtual void  dds::sub::DataReaderListener< T >::on_sample_lost (
              DataReader< T > & reader,
             const  dds::core::status::SampleLostStatus & status )  [pure virtual]
```

Handles the **dds::core::status::SampleLostStatus** (p. 2303) status.

Implemented in **dds::sub::NoOpDataReaderListener< RequestType >** (p. 1840), and **dds::sub::NoOpDataReaderListener< T >** (p. 1840).

8.57 rti::core::policy::DataReaderProtocol Class Reference

<<***extension***>> *(*p. *116)* Configures DataReader-specific aspects of the RTPS protocol

`#include <rti/core/policy/CorePolicy.hpp>`

Public Member Functions

- **DataReaderProtocol** ()

  Creates the default policy.
- **DataReaderProtocol** & **virtual_guid** (const **rti::core::Guid** &the_virtual_guid)

  <<***extension***>> *(*p. *116) The virtual GUID (Global Unique Identifier).*
- **rti::core::Guid virtual_guid** () const

  Getter (see setter with the same name)
- **DataReaderProtocol** & **rtps_object_id** (uint32_t the_rtps_object_id)

  <<***extension***>> *(*p. *116) The RTPS Object ID.*
- uint32_t **rtps_object_id** () const

  Getter (see setter with the same name)
- **DataReaderProtocol** & **expects_inline_qos** (bool the_expects_inline_qos)

  [DEPRECATED] (p. ***296)*** <<***extension***>> *(*p. *116) Specifies whether this DataReader expects inline QoS with every sample.*
- bool **expects_inline_qos** () const

  Getter (see setter with the same name)
- **DataReaderProtocol** & **disable_positive_acks** (bool the_disable_positive_acks)

  <<***extension***>> *(*p. *116) Whether the reader sends positive acknowledgements to writers.*
- bool **disable_positive_acks** () const

  Getter (see setter with the same name)
- **DataReaderProtocol** & **propagate_dispose_of_unregistered_instances** (bool the_propagate_dispose_of↩ _unregistered_instances)

  <<***extension***>> *(*p. *116) Indicates whether or not an instance can move to the* ***dds::sub::status::InstanceState↩ ::not_alive_disposed()*** *(*p. *1617) state without being in the* ***dds::sub::status::InstanceState::alive()*** *(*p. *1617) state.*
- bool **propagate_dispose_of_unregistered_instances** () const

  Getter (see setter with the same name)
- **DataReaderProtocol** & **propagate_unregister_of_disposed_instances** (bool the_propagate_unregister_of↩ _disposed_instances)

  <<***extension***>> *(*p. *116) Indicates whether or not an instance can move to the* ***dds::sub::status::InstanceState::not↩ _alive_no_writers()*** *(*p. *1617) state directly from the* ***dds::sub::status::InstanceState::not_alive_disposed()*** *(*p. *1617).*
- bool **propagate_unregister_of_disposed_instances** () const

  Getter (see setter with the same name)
- **DataReaderProtocol** & **rtps_reliable_reader** (const **rti::core::RtpsReliableReaderProtocol** &the_rtps_↩ reliable_reader)

  <<***extension***>> *(*p. *116) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a* ***dds::sub::DataReader*** *(*p. *976). This parameter only has effect if the reader is configured with* ***dds::core::policy::↩ ReliabilityKind_def::RELIABLE*** *(*p. *2157)* ***dds::core::policy::ReliabilityKind_def*** *(*p. *2156).*
- const **rti::core::RtpsReliableReaderProtocol** & **rtps_reliable_reader** () const

  Gets the reliable protocol settings by const reference (see setter)
- **rti::core::RtpsReliableReaderProtocol** & **rtps_reliable_reader** ()

  Gets the reliable protocol setting by reference (see setter)

8.57.1 Detailed Description

<<***extension***>> *(*p. *116)* Configures DataReader-specific aspects of the RTPS protocol

DDS has a standard protocol for packet (user and meta data) exchange between applications using DDS for communications. This QoS policy and **rti::core::policy::DataReaderProtocol** (p. 1059) give you control over configurable portions of the protocol, including the configuration of the reliable data delivery mechanism of the protocol on a per DataWriter or DataReader basis.

These configuration parameters control timing, timeouts, and give you the ability to tradeoff between speed of data loss detection and repair versus network and CPU bandwidth used to maintain reliability.

It is important to tune the reliability protocol (on a per **dds::pub::DataWriter** (p. 1138) and **dds::sub::DataReader** (p. 976) basis) to meet the requirements of the end-user application so that data can be sent between DataWriters and DataReaders in an efficient and optimal manner in the presence of data loss.

You can also use this QoS policy to control how RTI Connext responds to "slow" reliable DataReaders or ones that disconnect or are otherwise lost. See **dds::core::policy::Reliability** (p. 2149) for more information on the per-DataReader/DataWriter reliability configuration. **dds::core::policy::History** (p. 1602) and **dds::core::policy::ResourceLimits** (p. 2203) also play an important role in the DDS reliable protocol.

This QoS policy is an extension to the DDS standard.

Entity:

dds::sub::DataReader (p. 976)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.57.2 Constructor & Destructor Documentation

8.57.2.1 DataReaderProtocol()

```
rti::core::policy::DataReaderProtocol::DataReaderProtocol ( )  [inline]
```

Creates the default policy.

8.57.3 Member Function Documentation

8.57.3.1 virtual_guid() [1/2]

```
DataReaderProtocol & rti::core::policy::DataReaderProtocol::virtual_guid (
            const  rti::core::Guid & the_virtual_guid )
```

<<***extension***>> *(p. 116)* The virtual GUID (Global Unique Identifier).

The virtual GUID is used to uniquely identify different incarnations of the same **dds::sub::DataReader** (p. 976).

The association between a **dds::sub::DataReader** (p. 976) and its persisted state is done using the virtual GUID.

[default] rti::core::Guid::automatic() (p. 1597)

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288)

8.57.3.2 virtual_guid() [2/2]

```
rti::core::Guid rti::core::policy::DataReaderProtocol::virtual_guid ( ) const
```

Getter (see setter with the same name)

8.57.3.3 rtps_object_id() [1/2]

```
DataReaderProtocol & rti::core::policy::DataReaderProtocol::rtps_object_id (
            uint32_t the_rtps_object_id )
```

<<***extension***>> *(p. 116)* The RTPS Object ID.

This value is used to determine the RTPS object ID of a data reader according to the DDS-RTPS Interoperability Wire Protocol.

Only the last 3 bytes are used; the most significant byte is ignored.

If the default value is specified, RTI Connext will automatically assign the object ID based on a counter value (per participant) starting at 0x00800000. That value is incremented for each new data reader.

A rtps_object_id value in the interval [0x00800000,0x00ffffff] may collide with the automatic values assigned by RTI Connext. In those cases, the recomendation is not to use automatic object ID assigment.

[default] rti::core::policy::WireProtocol::RTPS_AUTO_ID (p. 2641)

[range] [0,0x00ffffff]

Categories

Immutable (p. 243)

8.57.3.4 rtps_object_id() [2/2]

```
uint32_t rti::core::policy::DataReaderProtocol::rtps_object_id ( ) const
```

Getter (see setter with the same name)

8.57.3.5 expects_inline_qos() [1/2]

```
DataReaderProtocol & rti::core::policy::DataReaderProtocol::expects_inline_qos (
            bool the_expects_inline_qos )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Specifies whether this DataReader expects inline QoS with every sample.

RTI Connext DataWriters do not match with DataReaders that set this field to true (because RTI Connext DataWriters do not support sending inline QoS), but here is how the field is meant to be used:

In RTI Connext, a **dds::sub::DataReader** (p. 976) nominally relies on **Discovery** (p. 1265) to propagate QoS on a matched **dds::pub::DataWriter** (p. 1138).

Alternatively, a **dds::sub::DataReader** (p. 976) may get information on a matched **dds::pub::DataWriter** (p. 1138) through QoS sent inline with a sample.

Asserting **rti::core::policy::DataReaderProtocol::expects_inline_qos** (p. 1062) indicates to a matching **dds::pub↩ ::DataWriter** (p. 1138) that this **dds::sub::DataReader** (p. 976) expects to receive inline QoS with every sample. The complete set of inline QoS that a **dds::pub::DataWriter** (p. 1138) may send inline is specified by the Real-Time Publish-Subscribe (RTPS) Wire Interoperability Protocol.

Because RTI Connext **dds::pub::DataWriter** (p. 1138) and **dds::sub::DataReader** (p. 976) cache **Discovery** (p. 1265) information, inline QoS are largely redundant and thus unecessary. Only for other stateless implementations whose **dds::sub::DataReader** (p. 976) does not cache **Discovery** (p. 1265) information is inline QoS necessary.

Also note that inline QoS are additional wire-payload that consume additional bandwidth and serialization and deserialization time.

[default] false

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288)

8.57.3.6 expects_inline_qos() [2/2]

```
bool rti::core::policy::DataReaderProtocol::expects_inline_qos ( ) const
```

Getter (see setter with the same name)

8.57.3.7 disable_positive_acks() [1/2]

```
DataReaderProtocol & rti::core::policy::DataReaderProtocol::disable_positive_acks (
            bool the_disable_positive_acks )
```

<<***extension***>> *(p. 116)* Whether the reader sends positive acknowledgements to writers.

If set to true, the reader does not send positive acknowledgments (ACKs) in response to Heartbeat messages. The reader will send negative acknowledgements (NACKs) when a Heartbeat advertises samples that it has not received.

Otherwise, if set to false (the default), the reader will send ACKs to writers that expect ACKs (**rti::core::policy::DataWriterProtocol::disable_positive_acks** (p. 1214) = false) and it will not send ACKs to writers that disable ACKs (**rti::core::policy::DataWriterProtocol::disable_positive_acks** (p. 1214) = true)

[default] false

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Reliability** (p. 222)

8.57.3.8 disable_positive_acks() [2/2]

```
bool rti::core::policy::DataReaderProtocol::disable_positive_acks ( ) const
```

Getter (see setter with the same name)

8.57.3.9 propagate_dispose_of_unregistered_instances() [1/2]

```
DataReaderProtocol & rti::core::policy::DataReaderProtocol::propagate_dispose_of_unregistered_
instances (
            bool the_propagate_dispose_of_unregistered_instances )
```

<<***extension***>> *(p. 116)* Indicates whether or not an instance can move to the **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) state without being in the **dds::sub::status::InstanceState::alive()** (p. 1617) state.

This field only applies to keyed readers.

When the field is set to true, the DataReader will receive dispose notifications even if the instance is not alive.

To guarantee the key availability through the usage of the API **dds::sub::DataReader::key_value** (p. 1003), this option should be used in combination with setting **rti::core::policy::DataWriterProtocol::serialize_key_with_dispose** (p. 1215) on the DataWriter to true.

[default] false

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193)

8.57.3.10 propagate_dispose_of_unregistered_instances() [2/2]

```
bool rti::core::policy::DataReaderProtocol::propagate_dispose_of_unregistered_instances ( ) const
```

Getter (see setter with the same name)

8.57.3.11 propagate_unregister_of_disposed_instances() [1/2]

```
DataReaderProtocol & rti::core::policy::DataReaderProtocol::propagate_unregister_of_disposed_↩
instances (
            bool the_propagate_unregister_of_disposed_instances )
```

<<***extension***>> *(*p. *116)* Indicates whether or not an instance can move to the **dds::sub::status::InstanceState↩ ::not_alive_no_writers()** (p. 1617) state directly from the **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617).

This field only applies to keyed readers.

When the field is set to true, the DataReader will receive unregister notifications even if the instance is not alive.

[default] false

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193)

8.57.3.12 propagate_unregister_of_disposed_instances() [2/2]

```
bool rti::core::policy::DataReaderProtocol::propagate_unregister_of_disposed_instances ( ) const
```

Getter (see setter with the same name)

8.57.3.13 rtps_reliable_reader() [1/3]

```
DataReaderProtocol & rti::core::policy::DataReaderProtocol::rtps_reliable_reader (
            const  rti::core::RtpsReliableReaderProtocol & the_rtps_reliable_reader )
```

<<***extension***>> *(*p. *116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **dds::sub::DataReader** (p. 976). This parameter only has effect if the reader is configured with **dds::core::policy::↩ ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core::policy::ReliabilityKind_def** (p. 2156).

For details, refer to the **rti::core::RtpsReliableReaderProtocol** (p. 2221)

[default] See **rti::core::RtpsReliableReaderProtocol** (p. 2221)

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.57.3.14 rtps_reliable_reader() [2/3]

```
const rti::core::RtpsReliableReaderProtocol & rti::core::policy::DataReaderProtocol::rtps_↩
reliable_reader ( ) const
```

Gets the reliable protocol settings by const reference (see setter)

8.57.3.15 rtps_reliable_reader() [3/3]

```
rti::core::RtpsReliableReaderProtocol & rti::core::policy::DataReaderProtocol::rtps_reliable_↩
reader ( )
```

Gets the reliable protocol setting by reference (see setter)

8.58 rti::core::status::DataReaderProtocolStatus Class Reference

<<***extension***>> *(*p. *354)* Information about the status **dds::core::status::StatusMask::datareader_protocol()** (p. 2387)

```
#include <Status.hpp>
```

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **EventCount64 received_sample_count** () const

  The number of samples received by a DataReader.
- **EventCount64 received_sample_bytes** () const

  The number of bytes received by a DataReader.
- **EventCount64 duplicate_sample_count** () const

  The number of samples from a remote DataWriter received, not for the first time, by a local DataReader.
- **EventCount64 duplicate_sample_bytes** () const

  The number of samples from a remote DataWriter received, not for the first time, by a local DataReader.
- **EventCount64 filtered_sample_count** () const

  [DEPRECATED]. *See:* ***rti::core::status::DataReaderCacheStatus::time_based_filter_dropped_sample_count*** *(*p. *1049)* ***rti::core::status::DataReaderCacheStatus::content_filter_dropped_sample_count*** *(*p. *1049)*
- **EventCount64 filtered_sample_bytes** () const

  [DEPRECATED]. *See:* ***rti::core::status::DataReaderCacheStatus::time_based_filter_dropped_sample_count*** *(*p. *1049)* ***rti::core::status::DataReaderCacheStatus::content_filter_dropped_sample_count*** *(*p. *1049)*
- **EventCount64 received_heartbeat_count** () const

  The number of Heartbeats from a remote DataWriter received by a local DataReader.
- **EventCount64 received_heartbeat_bytes** () const

  The number of bytes of Heartbeats from a remote DataWriter received by a local DataReader.
- **EventCount64 sent_ack_count** () const

The number of ACKs sent from a local DataReader to a matching remote DataWriter.

- **EventCount64 sent_ack_bytes** () const

  The number of bytes of ACKs sent from a local DataReader to a matching remote DataWriter.

- **EventCount64 sent_nack_count** () const

  The number of NACKs sent from a local DataReader to a matching remote DataWriter.

- **EventCount64 sent_nack_bytes** () const

  The number of bytes of NACKs sent from a local DataReader to a matching remote DataWriter.

- **EventCount64 received_gap_count** () const

  The number of GAPs received from remote DataWriter to this DataReader.

- **EventCount64 received_gap_bytes** () const

  The number of bytes of GAPs received from remote DataWriter to this DataReader.

- **EventCount64 rejected_sample_count** () const

  The number of times a sample is rejected because it cannot be accepted by a reliable DataReader.

- **rti::core::SequenceNumber first_available_sample_sequence_number** () const

  Sequence number of the first available sample in a matched DataWriters reliability queue.

- **rti::core::SequenceNumber last_available_sample_sequence_number** () const

  Sequence number of the last available sample in a matched Datawriter's reliability queue.

- **rti::core::SequenceNumber last_committed_sample_sequence_number** () const

  Sequence number of the newest sample received from the matched DataWriter committed to the DataReader's queue.

- int32_t **uncommitted_sample_count** () const

  Number of received samples that are not yet available to be read or taken, due to being received out of order.

- int64_t **received_fragment_count** () const

  The number of DATA_FRAG messages that have been received by this DataReader.

- int64_t **dropped_fragment_count** () const

  The number of DATA_FRAG messages that have been dropped by a DataReader.

- int64_t **reassembled_sample_count** () const

  The number of fragmented samples that have been reassembled by a DataReader.

- int64_t **sent_nack_fragment_count** () const

  The number of NACK fragments that have been sent from a DataReader to a DataWriter.

- int64_t **sent_nack_fragment_bytes** () const

  The number of NACK fragment bytes that have been sent from a DataReader to a DataWriter.

- int64_t **out_of_range_rejected_sample_count** () const

  The number of samples dropped by the DataReader due to received window is full and the sample is out-of-order.

8.58.1 Detailed Description

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::datareader_protocol()** (p. 2387)

Entity:

dds::sub::DataReader (p. 976)

8.58.2 Member Function Documentation

8.58.2.1 received_sample_count()

EventCount64 rti::core::status::DataReaderProtocolStatus::received_sample_count ( ) const [inline]

The number of samples received by a DataReader.

Depending on how the rti::core::status::DataReaderProtocol was obtained this may count samples coming from a specific DataWriter or from all the DataWriters that are matched with the DataReader.

If the rti::core::status::DataReaderProtocol is obtained using the **dds::sub::DataReader::datareader_protocol_status** (p. 1013) operation then it will count samples from any DataWriter. If the **DataReaderProtocolStatus** (p. 1065) is obtained using the **dds::sub::DataReader::matched_publication_datareader_protocol_status** (p. 1013) then it will count the samples for the DataWriter specified as a parameter to the function.

Duplicate samples arriving from the DataWriter(s) (e.g. via multiple network paths) are detected prior to increasing this counter. The duplicate samples are counted by rti::core::status::DataReaderProtocol::duplicate_sample_count.

If the DataReader has specified a ContentFilter the received samples that do not pass the filter are part of this counter. The filtered samples are counted by rti::core::status::DataReaderProtocol::filtered_sample_count.

Samples rejected because they do not fit on the DataReader Queue are also part of this counter.

Note the received_sample_count counts samples received from all DataWriters and it does not necessarily match the number of samples accepted into the DataReader Queue. This is because:

- Samples can also be inserted into the DataReader Queue by lifecycle events that are locally detected like an instance becoming not alive as a result of DataWriters leaving the network.
- Samples can be filtered out due to ContentFilter or TimeFilter
- Samples can be rejected because there is no space in DataReader Queue

Note that when data is fragmented, this statistic is updated when all of the fragments required to reassemble a sample are received, not when individual fragments are received.

8.58.2.2 received_sample_bytes()

EventCount64 rti::core::status::DataReaderProtocolStatus::received_sample_bytes ( ) const [inline]

The number of bytes received by a DataReader.

See also

rti::core::status::DataReaderProtocol::received_sample_count

Note that when data is fragmented, this statistic is updated upon the receipt of each fragment, not when a sample is reassembled.

8.58.2.3 duplicate_sample_count()

```
EventCount64 rti::core::status::DataReaderProtocolStatus::duplicate_sample_count ( ) const  [inline]
```

The number of samples from a remote DataWriter received, not for the first time, by a local DataReader.

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

8.58.2.4 duplicate_sample_bytes()

```
EventCount64 rti::core::status::DataReaderProtocolStatus::duplicate_sample_bytes ( ) const  [inline]
```

The number of samples from a remote DataWriter received, not for the first time, by a local DataReader.

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

8.58.2.5 filtered_sample_count()

```
EventCount64 rti::core::status::DataReaderProtocolStatus::filtered_sample_count ( ) const  [inline]
```

[DEPRECATED]. See: **rti::core::status::DataReaderCacheStatus::time_based_filter_dropped_sample_count** (p. 1049) **rti::core::status::DataReaderCacheStatus::content_filter_dropped_sample_count** (p. 1049)

8.58.2.6 filtered_sample_bytes()

```
EventCount64 rti::core::status::DataReaderProtocolStatus::filtered_sample_bytes ( ) const  [inline]
```

[DEPRECATED]. See: **rti::core::status::DataReaderCacheStatus::time_based_filter_dropped_sample_count** (p. 1049) **rti::core::status::DataReaderCacheStatus::content_filter_dropped_sample_count** (p. 1049)

8.58.2.7 received_heartbeat_count()

```
EventCount64 rti::core::status::DataReaderProtocolStatus::received_heartbeat_count ( ) const
[inline]
```

The number of Heartbeats from a remote DataWriter received by a local DataReader.

8.58.2.8 received_heartbeat_bytes()

EventCount64 rti::core::status::DataReaderProtocolStatus::received_heartbeat_bytes ( ) const [inline]

The number of bytes of Heartbeats from a remote DataWriter received by a local DataReader.

8.58.2.9 sent_ack_count()

EventCount64 rti::core::status::DataReaderProtocolStatus::sent_ack_count ( ) const [inline]

The number of ACKs sent from a local DataReader to a matching remote DataWriter.

8.58.2.10 sent_ack_bytes()

EventCount64 rti::core::status::DataReaderProtocolStatus::sent_ack_bytes ( ) const [inline]

The number of bytes of ACKs sent from a local DataReader to a matching remote DataWriter.

8.58.2.11 sent_nack_count()

EventCount64 rti::core::status::DataReaderProtocolStatus::sent_nack_count ( ) const [inline]

The number of NACKs sent from a local DataReader to a matching remote DataWriter.

8.58.2.12 sent_nack_bytes()

EventCount64 rti::core::status::DataReaderProtocolStatus::sent_nack_bytes ( ) const [inline]

The number of bytes of NACKs sent from a local DataReader to a matching remote DataWriter.

8.58.2.13 received_gap_count()

EventCount64 rti::core::status::DataReaderProtocolStatus::received_gap_count ( ) const [inline]

The number of GAPs received from remote DataWriter to this DataReader.

8.58.2.14 received_gap_bytes()

```
EventCount64 rti::core::status::DataReaderProtocolStatus::received_gap_bytes ( ) const  [inline]
```

The number of bytes of GAPs received from remote DataWriter to this DataReader.

8.58.2.15 rejected_sample_count()

```
EventCount64 rti::core::status::DataReaderProtocolStatus::rejected_sample_count ( ) const  [inline]
```

The number of times a sample is rejected because it cannot be accepted by a reliable DataReader.

Samples rejected by a reliable DataReader will be NACKed, and they will have to be resent by the DataWriter if they are still available in the DataWriter queue.

This counter will always be 0 when using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157).

8.58.2.16 first_available_sample_sequence_number()

```
rti::core::SequenceNumber rti::core::status::DataReaderProtocolStatus::first_available_sample_↩
sequence_number ( ) const  [inline]
```

Sequence number of the first available sample in a matched DataWriters reliability queue.

Applicable only for reliable DataReaders, and when retrieving matched DataWriter statuses.

Updated upon receiving Heartbeat submessages from a matched reliable DataWriter.

8.58.2.17 last_available_sample_sequence_number()

```
rti::core::SequenceNumber rti::core::status::DataReaderProtocolStatus::last_available_sample_↩
sequence_number ( ) const  [inline]
```

Sequence number of the last available sample in a matched Datawriter's reliability queue.

Applicable only for reliable DataReaders, and when retrieving matched DataWriter statuses.

Updated upon receiving Heartbeat submessages from a matched reliable DataWriter.

8.58.2.18 last_committed_sample_sequence_number()

```
rti::core::SequenceNumber rti::core::status::DataReaderProtocolStatus::last_committed_sample_↩
sequence_number ( ) const  [inline]
```

Sequence number of the newest sample received from the matched DataWriter committed to the DataReader's queue.

Applicable only when retrieving matched DataWriter statuses.

For best-effort DataReaders, this is the sequence number of the latest sample received.

For reliable DataReaders, this is the sequence number of the latest sample that is available to be read or taken from the DataReader's queue.

8.58.2.19 uncommitted_sample_count()

```
int32_t rti::core::status::DataReaderProtocolStatus::uncommitted_sample_count ( ) const  [inline]
```

Number of received samples that are not yet available to be read or taken, due to being received out of order.

Applicable only when retrieving matched DataWriter statuses.

8.58.2.20 received_fragment_count()

```
int64_t rti::core::status::DataReaderProtocolStatus::received_fragment_count ( ) const  [inline]
```

The number of DATA_FRAG messages that have been received by this DataReader.

This statistic is incremented upon the receipt of each DATA_FRAG message. Fragments from duplicate samples do not count towards this statistic. Applicable only when data is fragmented.

8.58.2.21 dropped_fragment_count()

```
int64_t rti::core::status::DataReaderProtocolStatus::dropped_fragment_count ( ) const  [inline]
```

The number of DATA_FRAG messages that have been dropped by a DataReader.

This statistic does not include malformed fragments. Applicable only when data is fragmented.

8.58.2.22 reassembled_sample_count()

```
int64_t rti::core::status::DataReaderProtocolStatus::reassembled_sample_count ( ) const  [inline]
```

The number of fragmented samples that have been reassembled by a DataReader.

This statistic is incremented when all of the fragments which are required to reassemble an entire sample have been received. Applicable only when data is fragmented.

8.58.2.23 sent_nack_fragment_count()

```
int64_t rti::core::status::DataReaderProtocolStatus::sent_nack_fragment_count ( ) const  [inline]
```

The number of NACK fragments that have been sent from a DataReader to a DataWriter.

NACK FRAG RTPS messages are sent when large data is used in conjunction with reliable communication. They have the same properties as NACK messages, but instead of applying to samples, they apply to fragments. Applicable only when data is fragmented.

8.58.2.24 sent_nack_fragment_bytes()

```
int64_t rti::core::status::DataReaderProtocolStatus::sent_nack_fragment_bytes ( ) const  [inline]
```

The number of NACK fragment bytes that have been sent from a DataReader to a DataWriter.

NACK FRAG RTPS messages are sent when large data is used in conjunction with reliable communication. They have the same properties as NACK messages, but instead of applying to samples, they apply to fragments. Applicable only when data is fragmented.

8.58.2.25 out_of_range_rejected_sample_count()

```
int64_t rti::core::status::DataReaderProtocolStatus::out_of_range_rejected_sample_count ( ) const
[inline]
```

The number of samples dropped by the DataReader due to received window is full and the sample is out-of-order.

When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157); if the DataReader received samples out-of-order, they are stored internally until the missing samples are received. The number of out-of-order samples that the DataReader can keep is set by **rti::core::RtpsReliableReaderProtocol::receive_window_size** (p. 2226). When the received window is full any out-of-order sample received will be dropped.

8.59 dds::sub::qos::DataReaderQos Class Reference

<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::sub::DataReader** (p. 976) supports

```
#include <dds/sub/qos/DataReaderQos.hpp>
```

Public Member Functions

- **DataReaderQos** ()

  *Creates a **DataReaderQos*** (p. *1072*) *with the default value for each policy.*
- **DataReaderQos** (const **dds::topic::qos::TopicQos** &topic_qos)

  *Creates a **DataReaderQos*** (p. *1072*) *with the given TopicQos.*
- **DataReaderQos** & **operator=** (const **dds::topic::qos::TopicQos** &topic_qos)

  *Copies into this **DataReaderQos*** (p. *1072*) *those policies that are also in TopicQos.*
- template<typename POLICY >
  const POLICY & **policy** () const

  Gets a QoS policy by const reference.
- template<typename POLICY >
  POLICY & **policy** ()

  Gets a QoS policy by reference.
- template<typename Policy >
  DataReaderQos & **policy** (const Policy &p)

  Sets a policy.
- template<typename Policy >
  DataReaderQos & **operator**<< (const Policy &p)

  Sets a policy.
- template<typename Policy >
  const **DataReaderQos** & **operator**>> (Policy &p) const

  Copies the values of a policy.

Related Functions

(Note that these are not member functions.)

- std::string **to_string** (const **DataReaderQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> (p. *354*) *Obtains a string representation of the **dds::sub::qos::DataReaderQos*** (p. *1072*)
- std::string **to_string** (const **DataReaderQos** &qos, const **DataReaderQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> (p. *354*) *Obtains a string representation of the **dds::sub::qos::DataReaderQos*** (p. *1072*)
- std::string **to_string** (const **DataReaderQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> (p. *354*) *Obtains a string representation of the **dds::sub::qos::DataReaderQos*** (p. *1072*)
- std::ostream & **operator**<< (std::ostream &out, const rti::sub::qos::DataReaderQos &qos)

  <<***extension***>> (p. *354*) *Prints a **dds::sub::qos::DataReaderQos*** (p. *1072*) *to an output stream.*

8.59.1 Detailed Description

<<***value-type***>> (p. *350*) Container of the QoS policies that a **dds::sub::DataReader** (p. 976) supports

8.59.2 DataReaderQos policies

A **DataReaderQos** (p. 1072) contains the following policies:

- **dds::core::policy::Durability** (p. 1433),
- **dds::core::policy::Deadline** (p. 1255),
- **dds::core::policy::LatencyBudget** (p. 1631),
- **dds::core::policy::Liveliness** (p. 1645),
- **dds::core::policy::Reliability** (p. 2149),
- **dds::core::policy::DestinationOrder** (p. 1258),
- **dds::core::policy::History** (p. 1602),
- **dds::core::policy::ResourceLimits** (p. 2203),
- **dds::core::policy::UserData** (p. 2591),
- **dds::core::policy::Ownership** (p. 1899),
- **dds::core::policy::TimeBasedFilter** (p. 2469),
- **dds::core::policy::ReaderDataLifecycle** (p. 2137),
- **dds::core::policy::DataRepresentation** (p. 1113),
- **dds::core::policy::TransportPriority** (p. 2552),
- **dds::core::policy::TypeConsistencyEnforcement** (p. 2563),
- **dds::core::policy::DataTag** (p. 1132),
- **rti::core::policy::DataReaderResourceLimits** (p. 1081),
- **rti::core::policy::DataReaderProtocol** (p. 1059),
- **rti::core::policy::TransportSelection** (p. 2554),
- **rti::core::policy::TransportUnicast** (p. 2556),
- **rti::core::policy::TransportMulticast** (p. 2543),
- **rti::core::policy::Property** (p. 1969),
- **rti::core::policy::Service** (p. 2350),
- **rti::core::policy::Availability** (p. 868),
- **rti::core::policy::EntityName** (p. 1530),
- **rti::core::policy::TypeSupport** (p. 2574)

To get or set policies use the **policy()** (p. 1077) getters and setters or operator << (see **examples** (p. 603)).

You must set certain members in a consistent manner:

dds::core::policy::Deadline (p. 1255) .period >= **dds::core::policy::TimeBasedFilter** (p. 2469) .minimum_↩ separation

dds::core::policy::History (p. 1602) .depth <= **dds::core::policy::ResourceLimits** (p. 2203) .max_samples_per_↩ instance

dds::core::policy::ResourceLimits (p. 2203) .max_samples_per_instance <= **dds::core::policy::ResourceLimits** (p. 2203) .max_samples **dds::core::policy::ResourceLimits** (p. 2203) .initial_samples <= **dds::core::policy::↩ ResourceLimits** (p. 2203) .max_samples

dds::core::policy::ResourceLimits (p. 2203) .initial_instances <= **dds::core::policy::ResourceLimits** (p. 2203) .max_instances

rti::core::policy::DataReaderResourceLimits (p. 1081) .initial_remote_writers_per_instance <= **rti::core::policy::↩ DataReaderResourceLimits** (p. 1081) .max_remote_writers_per_instance

rti::core::policy::DataReaderResourceLimits (p. 1081) .initial_infos <= **rti::core::policy::DataReaderResource↩ Limits** (p. 1081) .max_infos

rti::core::policy::DataReaderResourceLimits (p. 1081) .max_remote_writers_per_instance <= **rti::core::policy::↩ DataReaderResourceLimits** (p. 1081) .max_remote_writers

rti::core::policy::DataReaderResourceLimits (p. 1081) .max_samples_per_remote_writer <= **dds::core::policy::↩ ResourceLimits** (p. 2203) .max_samples

length of **dds::core::policy::UserData** (p. 2591) .value <= dds::domain::qos::DomainParticipantQos::resource_limits .reader_user_data_max_length

If any of the above are not true, **dds::sub::DataReader::qos(const dds::sub::qos::DataReaderQos&)** (p. 1008) will fail with **dds::core::InconsistentPolicyError** (p. 1610) and the **dds::sub::DataReader** (p. 976) constructors will fail with **dds::core::Error** (p. 1539)

See also

Qos Use Cases (p. 602)

Examples

Foo_subscriber.cxx.

8.59.3 Constructor & Destructor Documentation

8.59.3.1 DataReaderQos() [1/2]

```
dds::sub::qos::DataReaderQos::DataReaderQos ( )
```

Creates a **DataReaderQos** (p. 1072) with the default value for each policy.

Note

If you configure Qos in XML, obtain the default value from the default QosProvider, since it can be overriden by configuration:
```
DataReaderQos reader_qos = dds::core::QosProvider::Default().datareader_qos();
```

8.59.3.2 DataReaderQos() [2/2]

```
dds::sub::qos::DataReaderQos::DataReaderQos (
            const  dds::topic::qos::TopicQos & topic_qos )
```

Creates a **DataReaderQos** (p. 1072) with the given TopicQos.

A **dds::topic::qos::TopicQos** (p. 2509) contains a subset of the policies of a **DataReaderQos** (p. 1072). This constructor copies those common policies into this instance while initializing the policies that are only defined in **DataReaderQos** (p. 1072) to their default vaues.

Parameters

topic_qos	The TopicQos bring copied.

Example:
```
DatareaderQos reader_qos(topic.qos());
```

8.59.4 Member Function Documentation

8.59.4.1 operator=()

```
DataReaderQos & dds::sub::qos::DataReaderQos::operator= (
            const  dds::topic::qos::TopicQos & topic_qos )  [inline]
```

Copies into this **DataReaderQos** (p. 1072) those policies that are also in TopicQos.

A **dds::topic::qos::TopicQos** (p. 2509) contains a subset of the policies of a **DataReaderQos** (p. 1072). This assignment operator copies those common policies into this instance while leaving the policies that are only defined in **DataReaderQos** (p. 1072) unaltered.

Parameters

topic_qos	The TopicQos to copy the common policies from.

For example:

```
// Load a TopicQos from a Qos profile
TopicQos topic_qos = dds::core::QosProvider::Default().topic_qos("MyLibrary::MyProfile");
// Create a DataReaderQos with the default policies
DataReaderQos reader_qos = dds::core::QosProvider::Default().datareader_qos();
// Overwrite the policies that are also defined in topic_qos
reader_qos = topic_qos;
```

8.59.4.2 policy() [1/3]

```
template<typename POLICY >
const POLICY & dds::sub::qos::DataReaderQos::policy ( ) const
```

Gets a QoS policy by const reference.

Template Parameters

Policy	One of the **DataReaderQos policies** (p. 1074)

See also

Setting Qos Values (p. 603)

8.59.4.3 policy() [2/3]

```
template<typename POLICY >
POLICY & dds::sub::qos::DataReaderQos::policy ( )
```

Gets a QoS policy by reference.

Template Parameters

Policy	One of the **DataReaderQos policies** (p. 1074)

See also

Setting Qos Values (p. 603)

8.59.4.4 policy() [3/3]

```
template<typename Policy >
DataReaderQos & dds::sub::qos::DataReaderQos::policy (
            const Policy & p )  [inline]
```

Sets a policy.

See also

policy() (p. 1077)

Setting Qos Values (p. 603)

8.59.4.5 operator<<()

```
template<typename Policy >
DataReaderQos & dds::sub::qos::DataReaderQos::operator<< (
            const Policy & p )  [inline]
```

Sets a policy.

See also

policy() (p. 1077)

Setting Qos Values (p. 603)

8.59.4.6 operator>>()

```
template<typename Policy >
const  DataReaderQos & dds::sub::qos::DataReaderQos::operator>> (
            Policy & p ) const  [inline]
```

Copies the values of a policy.

Parameters

p	The destination where to copy the current value of the Policy

See also

policy() (p. 1077)

Setting Qos Values (p. 603)

8.59.5 Friends And Related Function Documentation

8.59.5.1 to_string() [1/3]

```
std::string to_string (
            const  DataReaderQos & qos,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::sub::qos::DataReaderQos** (p. 1072)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
DataReaderQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for DataReaderQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
DataReaderQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

8.59.5.2 to_string() [2/3]

```
std::string to_string (
            const  DataReaderQos & qos,
            const  DataReaderQos & base,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::sub::qos::DataReaderQos** (p. 1072)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

8.59.5.3 to_string() [3/3]

```
std::string to_string (
            const  DataReaderQos & qos,
            const  rti::core::qos_print_all_t & qos_print_all,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::sub::qos::DataReaderQos** (p. 1072)

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
```
to_string(qos, rti::core::qos_print_all);
```

Returns

The string representation of the qos

8.59.5.4 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const rti::sub::qos::DataReaderQos & qos )  [related]
```

<<***extension***>> *(*p. *354)* Prints a **dds::sub::qos::DataReaderQos** (p. 1072) to an output stream.

8.60 rti::core::policy::DataReaderResourceLimits Class Reference

<<***extension***>> *(*p. *116)* Configures the memory usage of a dds::pub::DataReader

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **DataReaderResourceLimits** ()

  *Creates a **DataReaderResourceLimits** (*p. *1081) qos policy with default values.*
- **DataReaderResourceLimits** & **max_remote_writers** (int32_t the_max_remote_writers)

  <<***extension***>> *(*p. *116) The maximum number of remote writers from which a **dds::sub::DataReader** (*p. *976) may read, including all instances.*
- int32_t **max_remote_writers** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_remote_writers_per_instance** (int32_t the_max_remote_writers_per_↩ instance)

  <<***extension***>> *(*p. *116) The maximum number of remote writers from which a **dds::sub::DataReader** (*p. *976) may read a single instance.*
- int32_t **max_remote_writers_per_instance** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_samples_per_remote_writer** (int32_t the_max_samples_per_remote↩ _writer)

  <<***extension***>> *(*p. *116) The maximum number of out-of-order samples from a given remote **dds::pub::DataWriter** (*p. *1138) that a **dds::sub::DataReader** (*p. *976) may store when maintaining a reliable connection to the **dds::pub::↩ DataWriter** (*p. *1138).*
- int32_t **max_samples_per_remote_writer** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_infos** (int32_t the_max_infos)

  <<***extension***>> *(*p. *116) The maximum number of info units that a **dds::sub::DataReader** (*p. *976) can use to store **dds::sub::SampleInfo** (*p. *2285).*
- int32_t **max_infos** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **initial_remote_writers** (int32_t the_initial_remote_writers)

  <<***extension***>> *(*p. *116) The initial number of remote writers from which a **dds::sub::DataReader** (*p. *976) may read, including all instances.*
- int32_t **initial_remote_writers** () const

  Getter (see setter with the same name)

- **DataReaderResourceLimits** & **initial_remote_writers_per_instance** (int32_t the_initial_remote_writers_per_instance)

  *<<**extension**>> (p. 116) The initial number of remote writers from which a **dds::sub::DataReader** (p. 976) may read a single instance.*
- int32_t **initial_remote_writers_per_instance** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **initial_infos** (int32_t the_initial_infos)

  *<<**extension**>> (p. 116) The initial number of info units that a **dds::sub::DataReader** (p. 976) can have, which are used to store **dds::sub::SampleInfo** (p. 2285).*
- int32_t **initial_infos** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **initial_outstanding_reads** (int32_t the_initial_outstanding_reads)

  *<<**extension**>> (p. 116) The initial number of outstanding calls to read/take (or one of their variants) on the same **dds::sub::DataReader** (p. 976) for which memory has not been returned by calling **dds::sub::LoanedSamples::return_loan()** (p. 1667).*
- int32_t **initial_outstanding_reads** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_outstanding_reads** (int32_t the_max_outstanding_reads)

  *<<**extension**>> (p. 116) The maximum number of outstanding read/take calls (or one of their variants) on the same **dds::sub::DataReader** (p. 976) for which memory has not been returned by calling **dds::sub::LoanedSamples::return_loan()** (p. 1667).*
- int32_t **max_outstanding_reads** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_samples_per_read** (int32_t the_max_samples_per_read)

  *<<**extension**>> (p. 116) The maximum number of data samples that the application can receive from the middleware in a single call to **dds::sub::DataReader::read** (p. 996) or **dds::sub::DataReader::take** (p. 997). If more data exists in the middleware, the application will need to issue multiple read/take calls.*
- int32_t **max_samples_per_read** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **disable_fragmentation_support** (bool the_disable_fragmentation_support)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Determines whether the **dds::sub::DataReader** (p. 976) can receive fragmented samples.*
- bool **disable_fragmentation_support** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_fragmented_samples** (int32_t the_max_fragmented_samples)

  *<<**extension**>> (p. 116) The maximum number of samples for which the **dds::sub::DataReader** (p. 976) may store fragments at a given point in time.*
- int32_t **max_fragmented_samples** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **initial_fragmented_samples** (int32_t the_initial_fragmented_samples)

  *<<**extension**>> (p. 116) The initial number of samples for which a **dds::sub::DataReader** (p. 976) may store fragments.*
- int32_t **initial_fragmented_samples** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_fragmented_samples_per_remote_writer** (int32_t the_max_fragmented_samples_per_remote_writer)

  *<<**extension**>> (p. 116) The maximum number of samples per remote writer for which a **dds::sub::DataReader** (p. 976) may store fragments.*
- int32_t **max_fragmented_samples_per_remote_writer** () const

  Getter (see setter with the same name)

- **DataReaderResourceLimits** & **max_fragments_per_sample** (int32_t the_max_fragments_per_sample)

  <<***extension***>> (p. *116) Maximum number of fragments for a single sample.*
- int32_t **max_fragments_per_sample** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **dynamically_allocate_fragmented_samples** (bool the_dynamically_↩ allocate_fragmented_samples)

  <<***extension***>> (p. *116) Determines whether the **dds::sub::DataReader*** (p. *976) pre-allocates storage for storing fragmented samples.*
- bool **dynamically_allocate_fragmented_samples** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_total_instances** (int32_t the_max_total_instances)

  <<***extension***>> (p. *116) Maximum number of instances for which a DataReader will keep state.*
- int32_t **max_total_instances** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_remote_virtual_writers** (int32_t the_max_remote_virtual_writers)

  <<***extension***>> (p. *116) The maximum number of remote virtual writers from which a **dds::sub::DataReader*** (p. *976) may read, including all instances.*
- int32_t **max_remote_virtual_writers** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **initial_remote_virtual_writers** (int32_t the_initial_remote_virtual_writers)

  <<***extension***>> (p. *116) The initial number of remote virtual writers from which a **dds::sub::DataReader*** (p. *976) may read, including all instances.*
- int32_t **initial_remote_virtual_writers** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_remote_virtual_writers_per_instance** (int32_t the_max_remote_↩ virtual_writers_per_instance)

  <<***extension***>> (p. *116) The maximum number of virtual remote writers that can be associated with an instance.*
- int32_t **max_remote_virtual_writers_per_instance** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **initial_remote_virtual_writers_per_instance** (int32_t the_initial_remote_↩ virtual_writers_per_instance)

  <<***extension***>> (p. *116) The initial number of virtual remote writers per instance.*
- int32_t **initial_remote_virtual_writers_per_instance** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_remote_writers_per_sample** (int32_t the_max_remote_writers_per_↩ sample)

  <<***extension***>> (p. *116) The maximum number of remote writers allowed to write the same sample.*
- int32_t **max_remote_writers_per_sample** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_query_condition_filters** (int32_t the_max_query_condition_filters)

  <<***extension***>> (p. *116) The maximum number of query condition filters a reader is allowed.*
- int32_t **max_query_condition_filters** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_app_ack_response_length** (int32_t the_max_app_ack_response_↩ length)

  <<***extension***>> (p. *116) Maximum length of application-level acknowledgment response data*
- int32_t **max_app_ack_response_length** () const

  Getter (see setter with the same name)

- **DataReaderResourceLimits** & **keep_minimum_state_for_instances** (bool the_keep_minimum_state_for_↩ instances)

  <<***extension***>> (p. *116) Whether or not keep a minimum instance state for up to* ***rti::core::policy::DataReader↩ ResourceLimits::max_total_instances*** *(p. 1096).*
- bool **keep_minimum_state_for_instances** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **initial_topic_queries** (int32_t the_initial_topic_queries)

  <<***extension***>> (p. *116) The initial number of TopicQueries allocated by a* ***dds::sub::DataReader*** *(p. 976).*
- int32_t **initial_topic_queries** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **max_topic_queries** (int32_t the_max_topic_queries)

  <<***extension***>> (p. *116) The maximum number of active TopicQueries that a* ***dds::sub::DataReader*** *(p. 976) can create.*
- int32_t **max_topic_queries** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **autopurge_remote_not_alive_writer_delay** (const **dds::core::Duration** &d)

  <<***extension***>> (p. *116) Maximum duration for which the* ***dds::sub::DataReader*** *(p. 976) will maintain information regarding a* ***dds::pub::DataWriter*** *(p. 1138) once the* ***dds::pub::DataWriter*** *(p. 1138) has become not alive.*
- **dds::core::Duration autopurge_remote_not_alive_writer_delay** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **autopurge_remote_virtual_writer_delay** (const **dds::core::Duration** &d)

  Maximum duration for which the ***dds::sub::DataReader*** *(p. 976) will maintain information regarding a virtual* ***dds::pub↩ ::DataWriter*** *(p. 1138) once the* ***dds::pub::DataWriter*** *(p. 1138) has become detached.*
- **dds::core::Duration autopurge_remote_virtual_writer_delay** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimits** & **lazily_allocate_batch_resources** (bool the_lazily_allocate_batch_resources)

  <<***extension***>> (p. *116) Determines whether the* ***dds::sub::DataReader*** *(p. 976) pre-allocates resources for receiving batches of samples or lazily allocates them when needed.*
- bool **lazily_allocate_batch_resources** () const

  Getter (see setter with the same name)
- const **AllocationSettings** & **shmem_ref_transfer_mode_attached_segment_allocation** () const

  <<***extension***>> (p. *116) Allocation resource for the shared memory segments attached by the* ***dds::sub::DataReader*** *(p. 976).*
- **AllocationSettings** & **shmem_ref_transfer_mode_attached_segment_allocation** ()

  Getter by non-const reference (see getter by const reference with the same name)
- const **DataReaderResourceLimitsInstanceReplacementSettings** & **instance_replacement** () const

  <<***extension***>> (p. *116) Sets the kind of instances allowed to be replaced for each instance state (****dds::sub::status↩ ::InstanceState*** *(p. 1615)) when a DataReader reaches* ***dds::core::policy::ResourceLimits::max_instances*** *(p. 2206).*
- **DataReaderResourceLimitsInstanceReplacementSettings** & **instance_replacement** ()

  Getter by non-const reference (see getter by const reference with the same name)

Static Public Member Functions

- static DDS_Long **auto_max_total_instances** ()

  <<***extension***>> (p. *116) This value is used to make* ***rti::core::policy::DataReaderResourceLimits::max_total_↩ instances*** *(p. 1096) equal to* ***dds::core::policy::ResourceLimits::max_instances*** *(p. 2206).*

8.60.1 Detailed Description

<<***extension***>> *(p. 116)* Configures the memory usage of a dds::pub::DataReader

DataReaders must allocate internal structures to handle the maximum number of DataWriters that may connect to it, whether or not a **dds::sub::DataReader** (p. 976) handles data fragmentation and how many data fragments that it may handle (for data samples larger than the MTU of the underlying network transport), how many simultaneous outstanding loans of internal memory holding data samples can be provided to user code, as well as others.

Most of these internal structures start at an initial size and, by default, will grow as needed by dynamically allocating additional memory. You may set fixed, maximum sizes for these internal structures if you want to bound the amount of memory that can be used by a **dds::sub::DataReader** (p. 976). By setting the initial size to the maximum size, you will prevent RTI Connext from dynamically allocating any memory after the creation of the **dds::sub::DataReader** (p. 976).

This QoS policy is an extension to the DDS standard.

Entity:

dds::sub::DataReader (p. 976)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.60.2 Constructor & Destructor Documentation

8.60.2.1 DataReaderResourceLimits()

```
rti::core::policy::DataReaderResourceLimits::DataReaderResourceLimits ( )  [inline]
```

Creates a **DataReaderResourceLimits** (p. 1081) qos policy with default values.

8.60.3 Member Function Documentation

8.60.3.1 max_remote_writers() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_remote_writers (
            int32_t the_max_remote_writers )
```

<<***extension***>> *(p. 116)* The maximum number of remote writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.

[default] dds::core::LENGTH_UNLIMITED (p. 438)
[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_remote_writers, >= max_remote_writers_per_instance

For unkeyed types, this value has to be equal to max_remote_writers_per_instance if max_remote_writers_per_instance is not equal to **dds::core::LENGTH_UNLIMITED** (p. 438).

Note: For efficiency, set `max_remote_writers >=` **rti::core::policy::DataReaderResourceLimits::max_remote_writers_per_instance** (p. 1086).

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.2 max_remote_writers() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_remote_writers ( ) const
```

Getter (see setter with the same name)

8.60.3.3 max_remote_writers_per_instance() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_remote_writers_per_instance (
            int32_t the_max_remote_writers_per_instance )
```

<<***extension***>> *(p. 116)* The maximum number of remote writers from which a **dds::sub::DataReader** (p. 976) may read a single instance.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1024] or **dds::core::LENGTH_UNLIMITED** (p. 438), <= max_remote_writers or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_remote_writers_per_instance

For unkeyed types, this value has to be equal to max_remote_writers if it is not **dds::core::LENGTH_UNLIMITED** (p. 438).

Note: For efficiency, set `max_remote_writers_per_instance <=` **rti::core::policy::DataReaderResourceLimits::max_remote_writers** (p. 1085)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.4 max_remote_writers_per_instance() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_remote_writers_per_instance ( ) const
```

Getter (see setter with the same name)

8.60.3.5 max_samples_per_remote_writer() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_samples_per_remote_writer (
            int32_t the_max_samples_per_remote_writer )
```

<<***extension***>> *(p. 116)* The maximum number of out-of-order samples from a given remote **dds::pub::DataWriter** (p. 1138) that a **dds::sub::DataReader** (p. 976) may store when maintaining a reliable connection to the **dds::pub::DataWriter** (p. 1138).

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 100 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), <= **dds::core::policy::ResourceLimits::max_samples** (p. 2205)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.6 max_samples_per_remote_writer() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_samples_per_remote_writer ( ) const
```

Getter (see setter with the same name)

References **rti::core::policy::DataWriterTransferMode::shmem_ref_settings()**.

8.60.3.7 max_infos() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_infos (
            int32_t the_max_infos )
```

<<***extension***>> *(p. 116)* The maximum number of info units that a **dds::sub::DataReader** (p. 976) can use to store **dds::sub::SampleInfo** (p. 2285).

When read/take is called on a DataReader, the DataReader passes a sequence of data samples and an associated sample info sequence. The sample info sequence contains additional information for each data sample.

max_infos determines the resources allocated for storing sample info. This memory is loaned to the application when passing a sample info sequence.

Note that sample info is a snapshot, generated when read/take is called.

max_infos should not be less than max_samples.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_infos

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.8 max_infos() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_infos ( ) const
```

Getter (see setter with the same name)

8.60.3.9 initial_remote_writers() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::initial_remote_writers (
            int32_t the_initial_remote_writers )
```

<<***extension***>> *(p. 116)* The initial number of remote writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.

[default] 2

[range] [1, 1 million], <= max_remote_writers

For unkeyed types this value has to be equal to initial_remote_writers_per_instance.

Note: For efficiency, set initial_remote_writers >= **rti::core::policy::DataReaderResourceLimits::initial_remote_writers_per_instance** (p. 1089).

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.10 initial_remote_writers() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::initial_remote_writers ( ) const
```

Getter (see setter with the same name)

8.60.3.11 initial_remote_writers_per_instance() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::initial_remote_writers_↩
per_instance (
            int32_t the_initial_remote_writers_per_instance )
```

<<***extension***>> *(*p. *116)* The initial number of remote writers from which a **dds::sub::DataReader** (p. 976) may read a single instance.

[default] 2

[range] [1,1024], <= max_remote_writers_per_instance

For unkeyed types this value has to be equal to initial_remote_writers.

Note: For efficiency, set initial_remote_writers_per_instance <= **rti::core::policy::DataReaderResourceLimits↩ ::initial_remote_writers** (p. 1088).

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.12 initial_remote_writers_per_instance() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::initial_remote_writers_per_instance ( ) const
```

Getter (see setter with the same name)

8.60.3.13 initial_infos() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::initial_infos (
            int32_t the_initial_infos )
```

<<***extension***>> *(p. 116)* The initial number of info units that a **dds::sub::DataReader** (p. 976) can have, which are used to store **dds::sub::SampleInfo** (p. 2285).

[default] 32

[range] [1,1 million], <= max_infos

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.14 initial_infos() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::initial_infos ( ) const
```

Getter (see setter with the same name)

8.60.3.15 initial_outstanding_reads() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::initial_outstanding_reads
(
            int32_t the_initial_outstanding_reads )
```

<<***extension***>> *(p. 116)* The initial number of outstanding calls to read/take (or one of their variants) on the same **dds::sub::DataReader** (p. 976) for which memory has not been returned by calling **dds::sub::LoanedSamples**↩ **::return_loan()** (p. 1667).

[default] 2

[range] [1, 65536], <= max_outstanding_reads

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.16 initial_outstanding_reads() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::initial_outstanding_reads ( ) const
```

Getter (see setter with the same name)

8.60.3.17 max_outstanding_reads() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_outstanding_reads (
            int32_t the_max_outstanding_reads )
```

<<***extension***>> *(p. 116)* The maximum number of outstanding read/take calls (or one of their variants) on the same **dds::sub::DataReader** (p. 976) for which memory has not been returned by calling **dds::sub::LoanedSamples↩ ::return_loan()** (p. 1667).

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 65536] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_outstanding_reads

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.18 max_outstanding_reads() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_outstanding_reads ( ) const
```

Getter (see setter with the same name)

8.60.3.19 max_samples_per_read() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_samples_per_read (
            int32_t the_max_samples_per_read )
```

<<***extension***>> *(p. 116)* The maximum number of data samples that the application can receive from the middleware in a single call to **dds::sub::DataReader::read** (p. 996) or **dds::sub::DataReader::take** (p. 997). If more data exists in the middleware, the application will need to issue multiple read/take calls.

When reading data using listeners, the expected number of samples available for delivery in a single `take` call is typically small: usually just one, in the case of unbatched data, or the number of samples in a single batch, in the case of batched data. (See **rti::core::policy::Batch** (p. 880) for more information about this feature.) When polling for data or using a **dds::core::cond::WaitSet** (p. 2617), however, multiple samples (or batches) could be retrieved at once, depending on the data rate.

A larger value for this parameter makes the API simpler to use at the expense of some additional memory consumption.

[default] 1024

[range] [1,65536]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.20 max_samples_per_read() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_samples_per_read ( ) const
```

Getter (see setter with the same name)

8.60.3.21 disable_fragmentation_support() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::disable_fragmentation_↩
support (
            bool the_disable_fragmentation_support )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) can receive fragmented samples.

When fragmentation support is not needed, disabling fragmentation support will save some memory resources.

[default] false

Categories

Immutable (p. 243), **Fragmentation** (p. 191), **Sending Large Data** (p. 197)

8.60.3.22 disable_fragmentation_support() [2/2]

```
bool rti::core::policy::DataReaderResourceLimits::disable_fragmentation_support ( ) const
```

Getter (see setter with the same name)

8.60.3.23 max_fragmented_samples() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_fragmented_samples (
            int32_t the_max_fragmented_samples )
```

<<***extension***>> *(p. 116)* The maximum number of samples for which the **dds::sub::DataReader** (p. 976) may store fragments at a given point in time.

At any given time, a **dds::sub::DataReader** (p. 976) may store fragments for up to `max_fragmented_samples` samples while waiting for the remaining fragments. These samples need not have consecutive sequence numbers and may have been sent by different **dds::pub::DataWriter** (p. 1138) instances.

Once all fragments of a sample have been received, the sample is treated as a regular sample and becomes subject to standard QoS settings such as **dds::core::policy::ResourceLimits::max_samples** (p. 2205).

The middleware will drop fragments if the `max_fragmented_samples` limit has been reached. For best-effort communication, the middleware will accept a fragment for a new sample, but drop the oldest fragmented sample from the same remote writer. For reliable communication, the middleware will drop fragments for any new samples until all fragments for at least one older sample from that writer have been received.

Only applies if **rti::core::policy::DataReaderResourceLimits::disable_fragmentation_support** (p. 1092) is false.

[default] 1024

[range] [1, 1 million]

Categories

> **Immutable** (p. 243), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Fragmentation** (p. 191), **Sending Large Data** (p. 197)

8.60.3.24 max_fragmented_samples() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_fragmented_samples ( ) const
```

Getter (see setter with the same name)

8.60.3.25 initial_fragmented_samples() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::initial_fragmented_↩
samples (
            int32_t the_initial_fragmented_samples )
```

<<***extension***>> *(p. 116)* The initial number of samples for which a **dds::sub::DataReader** (p. 976) may store fragments.

Only applies if **rti::core::policy::DataReaderResourceLimits::disable_fragmentation_support** (p. 1092) is false.

[default] 4

[range] [1,1024], <= max_fragmented_samples

Categories

> **Immutable** (p. 243), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Fragmentation** (p. 191), **Sending Large Data** (p. 197)

8.60.3.26 initial_fragmented_samples() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::initial_fragmented_samples ( ) const
```

Getter (see setter with the same name)

8.60.3.27 max_fragmented_samples_per_remote_writer() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_fragmented_samples_↩
per_remote_writer (
            int32_t the_max_fragmented_samples_per_remote_writer )
```

<<***extension***>> *(p. 116)* The maximum number of samples per remote writer for which a **dds::sub::DataReader** (p. 976) may store fragments.

Logical limit so a single remote writer cannot consume all available resources.

Only applies if **rti::core::policy::DataReaderResourceLimits::disable_fragmentation_support** (p. 1092) is false.

[default] 256

[range] [1, 1 million], <= max_fragmented_samples

Categories

Immutable (p. 243), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Fragmentation** (p. 191), **Sending Large Data** (p. 197)

8.60.3.28 max_fragmented_samples_per_remote_writer() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_fragmented_samples_per_remote_writer ( )
const
```

Getter (see setter with the same name)

8.60.3.29 max_fragments_per_sample() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_fragments_per_sample
(
            int32_t the_max_fragments_per_sample )
```

<<***extension***>> *(p. 116)* Maximum number of fragments for a single sample.

Only applies if **rti::core::policy::DataReaderResourceLimits::disable_fragmentation_support** (p. 1092) is false.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438)

Categories

Immutable (p. 243), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Fragmentation** (p. 191), **Sending Large Data** (p. 197)

8.60.3.30 max_fragments_per_sample() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_fragments_per_sample ( ) const
```

Getter (see setter with the same name)

8.60.3.31 dynamically_allocate_fragmented_samples() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::dynamically_allocate_↩
fragmented_samples (
            bool the_dynamically_allocate_fragmented_samples )
```

<<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) pre-allocates storage for storing fragmented samples.

By default, the middleware does not allocate memory upfront, but instead allocates memory from the heap upon receiving the first fragment of a new sample. The amount of memory allocated equals the amount of memory needed to store all fragments in the sample. Once all fragments of a sample have been received, the sample is deserialized and stored in the regular receive queue. At that time, the dynamically allocated memory is freed again.

This QoS setting is useful for large, but variable-sized data types where upfront memory allocation for multiple samples based on the maximum possible sample size may be expensive. The main disadvantage of not pre-allocating memory is that one can no longer guarantee the middleware will have sufficient resources at runtime.

If `dynamically_allocate_fragmented_samples` is set to false, the middleware will allocate memory upfront for storing fragments for up to **rti::core::policy::DataReaderResourceLimits::initial_fragmented_samples** (p. 1093) samples. This memory may grow up to **rti::core::policy::DataReaderResourceLimits::max_fragmented_samples** (p. 1092) if needed.

Only applies if **rti::core::policy::DataReaderResourceLimits::disable_fragmentation_support** (p. 1092) is false.

[default] true

Categories

Immutable (p. 243), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Fragmentation** (p. 191), **Sending Large Data** (p. 197)

8.60.3.32 dynamically_allocate_fragmented_samples() [2/2]

```
bool rti::core::policy::DataReaderResourceLimits::dynamically_allocate_fragmented_samples ( )
const
```

Getter (see setter with the same name)

8.60.3.33 max_total_instances() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_total_instances (
            int32_t the_max_total_instances )
```

<<***extension***>> *(*p. *116)* Maximum number of instances for which a DataReader will keep state.

The maximum number of instances actively managed by a DataReader is determined by **dds::core::policy::ResourceLimits::max_instances** (p. 2206).

These instances have associated DataWriters or samples in the DataReader's queue and are visible to the user through operations such as **dds::sub::DataReader::take** (p. 997), **dds::sub::DataReader::read** (p. 996), and **dds::sub::DataReader::key_value** (p. 1003).

The features Durable Reader State, **MultiChannel** (p. 1749) DataWriters and RTI Persistence **Service** (p. 2350) require RTI Connext to keep some internal state even for instances without DataWriters or samples in the DataReader's queue. The additional state is used to filter duplicate samples that could be coming from different DataWriter channels or from multiple executions of RTI Persistence **Service** (p. 2350).

The total maximum number of instances that will be managed by the middleware, including instances without associated DataWriters or samples, is determined by max_total_instances.

When a new instance is received, RTI Connext will check the resource limit **dds::core::policy::ResourceLimits::max_instances** (p. 2206). If the limit is exceeded, RTI Connext will drop the sample with the reason LOST_BY_INSTANCES_LIMIT. If the limit is not exceeded, RTI Connext will check max_total_instances. If max_total_instances is exceeded, RTI Connext will replace an existing instance without DataWriters and samples with the new one. The application could receive duplicate samples for the replaced instance if it becomes alive again.

The max_total_instances limit is not used if **rti::core::policy::DataReaderResourceLimits::keep_minimum_state_for_instances** (p. 1101) is false, and in that case should be left at the default value.

[default] rti::core::policy::DataReaderResourceLimits::auto_max_total_instances (p. 519)

[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438) or **rti::core::policy::DataReaderResourceLimits::auto_max_total_instances** (p. 519), >= **dds::core::policy::ResourceLimits::max_instances** (p. 2206)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Instance Lifecycle** (p. 193)

8.60.3.34 max_total_instances() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_total_instances ( ) const
```

Getter (see setter with the same name)

8.60.3.35 max_remote_virtual_writers() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_remote_virtual_writers (
            int32_t the_max_remote_virtual_writers )
```

<<***extension***>> *(*p. *116)* The maximum number of remote virtual writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.

When **dds::core::policy::Presentation::access_scope** (p. 1947) is set to **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951), this value determines the maximum number of DataWriter groups that can be managed by the **dds::sub::Subscriber** (p. 2410) containing this **dds::sub::DataReader** (p. 976).

Since the **dds::sub::Subscriber** (p. 2410) may contain more than one **dds::sub::DataReader** (p. 976), only the setting of the first applies.

[default] dds::core::LENGTH_UNLIMITED (p. 438)
[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_remote_virtual_writers, >= max_remote_virtual_writers_per_instance

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.36 max_remote_virtual_writers() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_remote_virtual_writers ( ) const
```

Getter (see setter with the same name)

8.60.3.37 initial_remote_virtual_writers() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::initial_remote_virtual_↩
writers (
            int32_t the_initial_remote_virtual_writers )
```

<<***extension***>> *(p. 116)* The initial number of remote virtual writers from which a **dds::sub::DataReader** (p. 976) may read, including all instances.

[default] 2
[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), <= max_remote_virtual_writers

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.38 initial_remote_virtual_writers() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::initial_remote_virtual_writers ( ) const
```

Getter (see setter with the same name)

8.60.3.39 max_remote_virtual_writers_per_instance() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_remote_virtual_↩
writers_per_instance (
            int32_t the_max_remote_virtual_writers_per_instance )
```

<<***extension***>> *(p. 116)* The maximum number of virtual remote writers that can be associated with an instance.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1024] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_remote_virtual_writers_per_instance

For unkeyed types, this value is ignored.

The features of Durable Reader State and **MultiChannel** (p. 1749) DataWriters, and RTI Persistence **Service** (p. 2350) require RTI Connext to keep some internal state per virtual writer and instance that is used to filter duplicate samples. These duplicate samples could be coming from different DataWriter channels or from multiple executions of RTI Persistence **Service** (p. 2350).

Once an association between a remote virtual writer and an instance is established, it is permanent – it will not disappear even if the physical writer incarnating the virtual writer is destroyed.

If max_remote_virtual_writers_per_instance is exceeded for an instance, RTI Connext will not associate this instance with new virtual writers. Duplicates samples from these virtual writers will not be filtered on the reader.

If you are not using Durable Reader State, **MultiChannel** (p. 1749) DataWriters or RTI Persistence **Service** (p. 2350) in your system, you can set this property to 1 to optimize resources.

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.40 max_remote_virtual_writers_per_instance() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_remote_virtual_writers_per_instance ( )
const
```

Getter (see setter with the same name)

8.60.3.41 initial_remote_virtual_writers_per_instance() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::initial_remote_virtual_↩
writers_per_instance (
            int32_t the_initial_remote_virtual_writers_per_instance )
```

<<***extension***>> *(*p. *116)* The initial number of virtual remote writers per instance.

[default] 2

[range] [1, 1024], <= max_remote_virtual_writers_per_instance

For unkeyed types, this value is ignored.

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.42 initial_remote_virtual_writers_per_instance() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::initial_remote_virtual_writers_per_instance (
) const
```

Getter (see setter with the same name)

8.60.3.43 max_remote_writers_per_sample() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_remote_writers_per_↩
sample (
            int32_t the_max_remote_writers_per_sample )
```

<<***extension***>> *(*p. *116)* The maximum number of remote writers allowed to write the same sample.

One scenario in which two DataWriters may write the same sample is Persistence **Service** (p. 2350). The DataReader may receive the same sample coming from the original DataWriter and from a Persistence **Service** (p. 2350) DataWriter.
[default] 3

[range] [1, 1024]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.44 max_remote_writers_per_sample() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_remote_writers_per_sample ( ) const
```

Getter (see setter with the same name)

8.60.3.45 max_query_condition_filters() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_query_condition_↩
filters (
            int32_t the_max_query_condition_filters )
```

<<***extension***>> *(*p. *116)* The maximum number of query condition filters a reader is allowed.

[default] 4

[range] [0, 32]

This value determines the maximum number of unique query condition content filters that a reader may create.

Each query condition content filter is comprised of both its `query_expression` and `query_parameters`. Two query conditions that have the same `query_expression` will require unique query condition filters if their `query_paramters` differ. Query conditions that differ only in their state masks will share the same query condition filter.

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Content Filtering** (p. 165)

8.60.3.46 max_query_condition_filters() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_query_condition_filters ( ) const
```

Getter (see setter with the same name)

8.60.3.47 max_app_ack_response_length() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_app_ack_response_↩
length (
            int32_t the_max_app_ack_response_length )
```

<<***extension***>> *(*p. *116)* Maximum length of application-level acknowledgment response data

The maximum length of response data in an application-level acknowledgment.

When set to zero, no response data is sent with application-level acknowledgments.

[default] 1

[range] [0, 32768]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Reliability** (p. 222), **Application Acknowledgement** (p. 158)

8.60.3.48 max_app_ack_response_length() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_app_ack_response_length ( ) const
```

Getter (see setter with the same name)

8.60.3.49 keep_minimum_state_for_instances() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::keep_minimum_state_for_↩
instances (
            bool the_keep_minimum_state_for_instances )
```

<<***extension***>> *(*p. *116)* Whether or not keep a minimum instance state for up to **rti::core::policy::DataReader↩ ResourceLimits::max_total_instances** (p. 1096).

The features Durable Reader State, multi-channel DataWriters, and Persistence **Service** (p. 2350) require RTI Connext to keep some minimal internal state even for instances without DataWriters or DDS samples in the DataReader's queue, or that have been purged due to a dispose. The additional state is used to filter duplicate DDS samples that could be coming from different DataWriter channels or from multiple executions of Persistence **Service** (p. 2350). The total maximum number of instances that will be managed by the middleware, including instances without associated Data↩ Writers or DDS samples or that have been purged due to a dispose, is determined by **rti::core::policy::DataReader↩ ResourceLimits::max_total_instances** (p. 1096).

This additional state will only be kept for up to max_total_instances if this field is set to true, otherwise the additional state will not be kept for any instances.

The minimum state includes information such as the source timestamp of the last sample received by the instance and the last sequence number received from a virtual GUID.

[default] true

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Instance Lifecycle** (p. 193)

8.60.3.50 keep_minimum_state_for_instances() [2/2]

```
bool rti::core::policy::DataReaderResourceLimits::keep_minimum_state_for_instances ( ) const
```

Getter (see setter with the same name)

8.60.3.51 initial_topic_queries() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::initial_topic_queries (
            int32_t the_initial_topic_queries )
```

<<***extension***>> *(*p. *116)* The initial number of TopicQueries allocated by a **dds::sub::DataReader** (p. 976).

[default] 1

See also

rti::sub::TopicQuery (p. 2516)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Topic Queries** (p. 239)

8.60.3.52 initial_topic_queries() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::initial_topic_queries ( ) const
```

Getter (see setter with the same name)

8.60.3.53 max_topic_queries() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::max_topic_queries (
            int32_t the_max_topic_queries )
```

<<***extension***>> *(*p. *116)* The maximum number of active TopicQueries that a **dds::sub::DataReader** (p. 976) can create.

Once this limit is reached, a **dds::sub::DataReader** (p. 976) can create more TopicQueries only if it deletes some of the previously created ones.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

See also

rti::sub::TopicQuery (p. 2516)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Topic Queries** (p. 239)

8.60.3.54 max_topic_queries() [2/2]

```
int32_t rti::core::policy::DataReaderResourceLimits::max_topic_queries ( ) const
```

Getter (see setter with the same name)

8.60.3.55 autopurge_remote_not_alive_writer_delay() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::autopurge_remote_not_↩
alive_writer_delay (
            const  dds::core::Duration & d )
```

<<***extension***>> *(p. 116)* Maximum duration for which the **dds::sub::DataReader** (p. 976) will maintain information regarding a **dds::pub::DataWriter** (p. 1138) once the **dds::pub::DataWriter** (p. 1138) has become not alive.

After this time elapses, the **dds::sub::DataReader** (p. 976) will purge all internal information regarding the not alive **dds::pub::DataWriter** (p. 1138).

See also

dds::core::policy::Liveliness (p. 1645) for more information on when a **dds::pub::DataWriter** (p. 1138) is considered not alive.

When set to **dds::core::Duration::automatic()** (p. 1451), this parameter is set to 10 times the value of **rti::core↩ ::policy::DiscoveryConfig::participant_liveliness_lease_duration** (p. 1279).

This QoS only applies when the **dds::sub::DataReader** (p. 976) is using **rti::core::policy::InstanceState↩ ConsistencyKind::recover_state** (p. 545) for the **dds::core::policy::Reliability::instance_state_consistency_kind** (p. 2155) QoS. When using **rti::core::policy::InstanceStateConsistencyKind::recover_state** (p. 545), a **dds::sub↩ ::DataReader** (p. 976) keeps state about all **dds::pub::DataWriter** (p. 1138) entities and the instances they were writing in order to be able to transition those instances back to their correct state if liveliness with the **dds::pub::DataWriter** (p. 1138) is recovered. This can cause unbounded memory growth if that state is never purged and **dds::pub::Data↩ Writer** (p. 1138) entities continuously come and go in a system. This QoS avoids that unbounded memory growth by setting a time at which that state will be purged.

This QoS should be set such that it is longer than the longest period of time for which a **dds::pub::DataWriter** (p. 1138) and **dds::sub::DataReader** (p. 976) are expected to be disconnected and then reconnected in your system.

An alternative to using this QoS to purge the state is to set the **rti::core::policy::DataReaderResourceLimits::max↩ _remote_writers** (p. 1085) QoS to a finite value. If that QoS is set to a finite value and the number of alive + not alive **dds::pub::DataWriter** (p. 1138) entities reaches the limit when a new **dds::pub::DataWriter** (p. 1138) is discovered, the oldest not alive **dds::pub::DataWriter** (p. 1138) will be replaced.

[default] dds::core::Duration::automatic() (p. 1451)

[range] $>$ 0 or **dds::core::Duration::automatic()** (p. 1451)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Instance Lifecycle** (p. 193), **Liveliness** (p. 199)

8.60.3.56 autopurge_remote_not_alive_writer_delay() [2/2]

```
dds::core::Duration rti::core::policy::DataReaderResourceLimits::autopurge_remote_not_alive_↩
writer_delay ( ) const
```

Getter (see setter with the same name)

8.60.3.57 autopurge_remote_virtual_writer_delay() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::autopurge_remote_virtual↩
_writer_delay (
            const  dds::core::Duration & d )
```

Maximum duration for which the **dds::sub::DataReader** (p. 976) will maintain information regarding a virtual **dds::pub::DataWriter** (p. 1138) once the **dds::pub::DataWriter** (p. 1138) has become detached.

Determines how long the **dds::sub::DataReader** (p. 976) will maintain information regarding a virtual **dds::pub::DataWriter** (p. 1138) that has has become detached.

After this time elapses, the **dds::sub::DataReader** (p. 976) will purge all internal information regarding the **dds::pub::DataWriter** (p. 1138).

[default] dds::core::Duration::infinite() (p. 1451)

[range] >= 0 or **dds::core::Duration::infinite()** (p. 1451)

8.60.3.58 autopurge_remote_virtual_writer_delay() [2/2]

```
dds::core::Duration rti::core::policy::DataReaderResourceLimits::autopurge_remote_virtual_↩
writer_delay ( ) const
```

Getter (see setter with the same name)

8.60.3.59 lazily_allocate_batch_resources() [1/2]

```
DataReaderResourceLimits & rti::core::policy::DataReaderResourceLimits::lazily_allocate_batch_↩
resources (
            bool the_lazily_allocate_batch_resources )
```

<<***extension***>> *(p. 116)* Determines whether the **dds::sub::DataReader** (p. 976) pre-allocates resources for receiving batches of samples or lazily allocates them when needed.

By default, the middleware does not allocate memory up front to manage the batches of samples received, but instead allocates memory from the heap upon receiving the first batch of samples.

Lazily allocating batching resources helps reduce memory consumption when batches are not used, but it may introduce some latency when receiving the first batch of samples, since memory needs to be allocated from the heap. If you know that your application will be using batches, you may set this value to false if you are concerned about that latency.

[default] true

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.60.3.60 lazily_allocate_batch_resources() [2/2]

```
bool rti::core::policy::DataReaderResourceLimits::lazily_allocate_batch_resources ( ) const
```

Getter (see setter with the same name)

8.60.3.61 shmem_ref_transfer_mode_attached_segment_allocation() [1/2]

```
const AllocationSettings & rti::core::policy::DataReaderResourceLimits::shmem_ref_transfer_mode↩
_attached_segment_allocation ( ) const
```

<<***extension***>> *(*p. *116)* Allocation resource for the shared memory segments attached by the **dds::sub::Data↩ Reader** (p. 976).

The max_count does not limit the total number of shared memory segments used by the **dds::sub::DataReader** (p. 976). When this limit is hit, the **dds::sub::DataReader** (p. 976) will try to detach from a segment that doesn't contain any loaned samples and attach to a new segment. If samples are loaned from all attached segments, then the **dds::sub::DataReader** (p. 976) will fail to attach to the new segment. This scenario will result in a sample loss.

[default] initial_count = **rti::core::AllocationSettings::AUTO_COUNT** (p. 805) (**rti::core::policy::DataReader↩ ResourceLimits::initial_remote_writers** (p. 1088)); max_count = **rti::core::AllocationSettings::AUTO_COUNT** (p. 805) (**rti::core::policy::DataReaderResourceLimits::max_remote_writers** (p. 1085)); incremental_count = **rti↩ ::core::AllocationSettings::AUTO_COUNT** (p. 805) (0 if initial_count = max_count; -1 otherwise);

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Sending Large Data** (p. 197)

8.60.3.62 shmem_ref_transfer_mode_attached_segment_allocation() [2/2]

```
AllocationSettings & rti::core::policy::DataReaderResourceLimits::shmem_ref_transfer_mode_↩
attached_segment_allocation ( )
```

Getter by non-const reference (see getter by const reference with the same name)

8.60.3.63 instance_replacement() [1/2]

```
const DataReaderResourceLimitsInstanceReplacementSettings & rti::core::policy::DataReaderResource↩
Limits::instance_replacement ( ) const
```

<<***extension***>> *(p. 116)* Sets the kind of instances allowed to be replaced for each instance state (**dds::sub↩ ::status::InstanceState** (p. 1615)) when a DataReader reaches **dds::core::policy::ResourceLimits::max_instances** (p. 2206).

When **dds::core::policy::ResourceLimits::max_instances** (p. 2206) is reached, a **dds::sub::DataReader** (p. 976) will try to make room for a new instance by attempting to reclaim an existing instance based on the instance replacement kinds specified by this field.

A DataReader can choose what kinds of instances can be replaced for each **dds::sub::status::InstanceState** (p. 1615) separately. This means, for example, that a DataReader can choose to not allow replacing alive (**dds::sub::status↩ ::InstanceState::alive()** (p. 1617)) instances but allow replacement of empty disposed (**dds::sub::status::Instance↩ State::not_alive_disposed()** (p. 1617)) instances.

Only instances whose states match the specified kinds are eligible to be replaced. In addition, there must be no outstanding loans on any of the samples belonging to the instance for it to be considered for replacement.

For all kinds, a **dds::sub::DataReader** (p. 976) will replace the least-recently-updated instance satisfying that kind. An instance is considered 'updated' when a valid sample or dispose sample is received and accepted for that instance. When using **dds::core::policy::OwnershipKind_def::EXCLUSIVE** (p. 1906), only samples that are received from the owner of the instance will cause the instance to be considered updated. An instance is not considered updated when an unregister sample is received because the unregister message simply indicates that there is one less writer that has updates for the instance, not that the instance itself was updated.

If no replaceable instance exists, the sample for the new instance will be considered lost with lost reason **rti::core↩ ::status::SampleLostState::lost_by_instances_limit()** (p. 2299) and the instance will not be asserted into the Data↩ Reader queue.

[default]

- **rti::core::DataReaderResourceLimitsInstanceReplacementSettings::alive_instance_removal** (p. 1110) = **rti::core::policy::DataReaderInstanceRemovalKind_def::NO_INSTANCE** (p. 1054)
- **rti::core::DataReaderResourceLimitsInstanceReplacementSettings::disposed_instance_removal** (p. 1110) = **rti::core::policy::DataReaderInstanceRemovalKind_def::EMPTY_INSTANCES** (p. 1054)
- **rti::core::DataReaderResourceLimitsInstanceReplacementSettings::no_writers_instance_removal** (p. 1110) = **rti::core::policy::DataReaderInstanceRemovalKind_def::EMPTY_INSTANCES** (p. 1054)

See also

rti::core::DataReaderResourceLimitsInstanceReplacementSettings (p. 1108)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Instance Lifecycle** (p. 193)

8.60.3.64 instance_replacement() [2/2]

DataReaderResourceLimitsInstanceReplacementSettings & rti::core::policy::DataReaderResource↩
Limits::instance_replacement ( )

Getter by non-const reference (see getter by const reference with the same name)

8.61 rti::core::DataReaderResourceLimitsInstanceReplacementSettings Class Reference

<<***extension***>> *(*p. *354)* How instances are replaced in the DataReader queue when resource limits are reached.

#include <rti/core/PolicySettings.hpp>

Public Member Functions

- **DataReaderResourceLimitsInstanceReplacementSettings** ()

  Creates an instance with the default removal kind for each instance state: ***rti::core::policy::DataReaderInstance↩ RemovalKind_def::NO_INSTANCE*** *(*p. *1054) for alive instances and* ***rti::core::policy::DataReaderInstanceRemoval↩ Kind_def::EMPTY_INSTANCES*** *(*p. *1054) for no_writers and dispose instances.*
- **DataReaderResourceLimitsInstanceReplacementSettings** (**rti::core::policy::DataReaderInstance↩ RemovalKind** the_alive_instance_removal, **rti::core::policy::DataReaderInstanceRemovalKind** the_↩ disposed_instance_removal, **rti::core::policy::DataReaderInstanceRemovalKind** the_no_writers_instance↩ _removal)

  Creates an instance with the given ***rti::core::policy::DataReaderInstanceRemovalKind_def*** *(*p. *1053) for alive, disposed and no_writers instances, respectively.*
- **rti::core::policy::DataReaderInstanceRemovalKind alive_instance_removal** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimitsInstanceReplacementSettings** & **alive_instance_removal** (**rti::core::policy↩ ::DataReaderInstanceRemovalKind** the_alive_instance_removal)

  Sets the instance replacement policy for instances in the ***dds::sub::status::InstanceState::alive()*** *(*p. *1617) state.*
- **rti::core::policy::DataReaderInstanceRemovalKind disposed_instance_removal** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimitsInstanceReplacementSettings** & **disposed_instance_removal** (**rti::core↩ ::policy::DataReaderInstanceRemovalKind** the_disposed_instance_removal)

  Sets the instance replacement policy for instances in the ***dds::sub::status::InstanceState::not_alive_disposed()*** *(*p. *1617) state.*
- **rti::core::policy::DataReaderInstanceRemovalKind no_writers_instance_removal** () const

  Getter (see setter with the same name)
- **DataReaderResourceLimitsInstanceReplacementSettings** & **no_writers_instance_removal** (**rti::core↩ ::policy::DataReaderInstanceRemovalKind** the_no_writers_instance_removal)

  Sets the instance replacement policy for instances in the ***dds::sub::status::InstanceState::not_alive_no_writers()*** *(*p. *1617) state.*

8.61.1 Detailed Description

<<***extension***>> *(p. 354)* How instances are replaced in the DataReader queue when resource limits are reached.

[default]

- **rti::core::DataReaderResourceLimitsInstanceReplacementSettings::alive_instance_removal** (p. 1110) = **rti::core::policy::DataReaderInstanceRemovalKind_def::NO_INSTANCE** (p. 1054)
- **rti::core::DataReaderResourceLimitsInstanceReplacementSettings::disposed_instance_removal** (p. 1110) = **rti::core::policy::DataReaderInstanceRemovalKind_def::EMPTY_INSTANCES** (p. 1054)
- **rti::core::DataReaderResourceLimitsInstanceReplacementSettings::no_writers_instance_removal** (p. 1110) = **rti::core::policy::DataReaderInstanceRemovalKind_def::EMPTY_INSTANCES** (p. 1054)

See also

rti::core::policy::DataReaderResourceLimits::instance_replacement (p. 1106)

8.61.2 Constructor & Destructor Documentation

8.61.2.1 DataReaderResourceLimitsInstanceReplacementSettings() [1/2]

```
rti::core::DataReaderResourceLimitsInstanceReplacementSettings::DataReaderResourceLimitsInstance↩
ReplacementSettings ( )  [inline]
```

Creates an instance with the default removal kind for each instance state: **rti::core::policy::DataReaderInstance↩ RemovalKind_def::NO_INSTANCE** (p. 1054) for alive instances and **rti::core::policy::DataReaderInstance↩ RemovalKind_def::EMPTY_INSTANCES** (p. 1054) for no_writers and dispose instances.

8.61.2.2 DataReaderResourceLimitsInstanceReplacementSettings() [2/2]

```
rti::core::DataReaderResourceLimitsInstanceReplacementSettings::DataReaderResourceLimitsInstance↩
ReplacementSettings (
              rti::core::policy::DataReaderInstanceRemovalKind the_alive_instance_removal,
              rti::core::policy::DataReaderInstanceRemovalKind the_disposed_instance_removal,
              rti::core::policy::DataReaderInstanceRemovalKind the_no_writers_instance_removal )
```

Creates an instance with the given **rti::core::policy::DataReaderInstanceRemovalKind_def** (p. 1053) for alive, disposed and no_writers instances, respectively.

8.61.3 Member Function Documentation

8.61.3.1 alive_instance_removal() [1/2]

```
rti::core::policy::DataReaderInstanceRemovalKind rti::core::DataReaderResourceLimitsInstance↩
ReplacementSettings::alive_instance_removal ( ) const
```

Getter (see setter with the same name)

8.61.3.2 alive_instance_removal() [2/2]

```
DataReaderResourceLimitsInstanceReplacementSettings & rti::core::DataReaderResourceLimitsInstance↩
ReplacementSettings::alive_instance_removal (
            rti::core::policy::DataReaderInstanceRemovalKind the_alive_instance_removal )
```

Sets the instance replacement policy for instances in the **dds::sub::status::InstanceState::alive()** (p. 1617) state.

[default] rti::core::policy::DataReaderInstanceRemovalKind_def::NO_INSTANCE (p. 1054)

8.61.3.3 disposed_instance_removal() [1/2]

```
rti::core::policy::DataReaderInstanceRemovalKind rti::core::DataReaderResourceLimitsInstance↩
ReplacementSettings::disposed_instance_removal ( ) const
```

Getter (see setter with the same name)

8.61.3.4 disposed_instance_removal() [2/2]

```
DataReaderResourceLimitsInstanceReplacementSettings & rti::core::DataReaderResourceLimitsInstance↩
ReplacementSettings::disposed_instance_removal (
            rti::core::policy::DataReaderInstanceRemovalKind the_disposed_instance_removal )
```

Sets the instance replacement policy for instances in the **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) state.

[default] rti::core::policy::DataReaderInstanceRemovalKind_def::EMPTY_INSTANCES (p. 1054)

8.61.3.5 no_writers_instance_removal() [1/2]

```
rti::core::policy::DataReaderInstanceRemovalKind rti::core::DataReaderResourceLimitsInstance↩
ReplacementSettings::no_writers_instance_removal ( ) const
```

Getter (see setter with the same name)

8.61.3.6 no_writers_instance_removal() [2/2]

```
DataReaderResourceLimitsInstanceReplacementSettings & rti::core::DataReaderResourceLimitsInstance↩
ReplacementSettings::no_writers_instance_removal (
            rti::core::policy::DataReaderInstanceRemovalKind the_no_writers_instance_removal )
```

Sets the instance replacement policy for instances in the **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) state.

[default] rti::core::policy::DataReaderInstanceRemovalKind_def::EMPTY_INSTANCES (p. 1054)

8.62 rti::sub::cond::DataReaderStatusConditionHandler< T > Class Template Reference

Realization of a **functor handler** (p. 2134) that handles the status of a **dds::sub::DataReader** (p. 976).

```
#include <DataReaderStatusConditionHandler.hpp>
```

Public Member Functions

- **DataReaderStatusConditionHandler** (**dds::sub::DataReader**< T > &reader, **dds::sub::DataReader↩ Listener**< T > *listener, const **dds::core::status::StatusMask** &listener_mask)

  Creates a new rti::core::sub::DataReaderStatusConditionHandler instance.

8.62.1 Detailed Description

template<**typename T**>
class rti::sub::cond::DataReaderStatusConditionHandler< T >

Realization of a **functor handler** (p. 2134) that handles the status of a **dds::sub::DataReader** (p. 976).

A rti::core::sub::DataReaderStatusConditionHandler demultiplexes a **dds::sub::DataReader** (p. 976) status change into the corresponding callback of a provided **dds::sub::DataReaderListener** (p. 1055) implementation.

Note that the **dds::sub::DataReaderListener** (p. 1055) notifications have different considerations than if the were made by the **dds::sub::DataReader** (p. 976) directly:

- Context: The **dds::sub::DataReaderListener** (p. 1055) callback context is the one of the thread that dispatches the **dds::core::cond::Condition** (p. 948) where this handler is set. For instance, if you attach the condition to a **rti::core::cond::AsyncWaitSet** (p. 840), the context will be one of the threads within the pool.
- Status clearing: All the **dds::sub::DataReader** (p. 976)'s enabled statuses are cleared upon condition dispatch except the **dds::core::status::StatusMask::data_available()** (p. 2383), which will not be cleared until your application reads the data.
- Exclusive Area: Restrictions depend on the context of the dipsatching thread. For instance, if the **rti::core::cond::AsyncWaitSet** (p. 840) dispatches the condition, the listener notications are free of any exclusive area restrictions.

The rti::core::sub::DataReaderStatusConditionHandler is a convenience to handle the status changes of a **dds::sub::DataReader** (p. 976). You can install a rti::core::sub::DataReaderStatusConditionHandler as the handler of a reader's **dds::core::cond::StatusCondition** (p. 2372). You can then attach it to a **dds::core::cond::WaitSet** (p. 2617) or **rti::core::cond::AsyncWaitSet** (p. 840) and receive status changes notifications through a specific **dds::sub::DataReaderListener** (p. 1055) implementation instance.

8.62.2 Constructor & Destructor Documentation

8.62.2.1 DataReaderStatusConditionHandler()

```
template<typename T >
rti::sub::cond::DataReaderStatusConditionHandler< T >::DataReaderStatusConditionHandler (
              dds::sub::DataReader< T > & reader,
              dds::sub::DataReaderListener< T > * listener,
             const  dds::core::status::StatusMask & listener_mask )  [inline]
```

Creates a new rti::core::sub::DataReaderStatusConditionHandler instance.

The created **DataReaderStatusConditionHandler** (p. 1111) can set as **functor handler** (p. 2134) in any **dds::core::cond::Condition** (p. 948) and will demultiplex the specified status changes from the specified **dds::sub::DataReader** (p. 976)

Parameters

reader	<<***in***>> *(p. 355)* The **dds::sub::DataReader** (p. 976) for which the status changes are demultiplexed to the specified `listener`
listener	<<***in***>> *(p. 355)* that receives the status changes notifications from the specified `reader`.
listener_mask	<<***in***>> *(p. 355)* Specifies which status changes from `the` reader to demultiplex to the `listener`.

See also

dds::sub::DataReader (p. 976)

dds::sub::DataReader::set_listener (p. 1006)

dds::core::cond::StatusCondition (p. 2372)

functor handler (p. 2134)

8.63 dds::core::policy::DataRepresentation Class Reference

Contains the data representations supported by entities.

`#include <dds/core/policy/CorePolicy.hpp>`

Public Member Functions

- **DataRepresentation** ()

  Creates an instance with one element, ***auto_id()*** *(p. 521).*
- **DataRepresentation** (const **DataRepresentationIdSeq** &seq)

  Creates an instance with a sequence of DataRepresentationId.
- **DataRepresentation** (const **DataRepresentationId** *value_begin, const **DataRepresentationId** *value_end)

  Creates an instance with a sequence of DataRepresentationId.
- template<typename ShortIter >
  DataRepresentation & **value** (ShortIter the_begin, ShortIter the_end)

  Sets the DataRepresentationId sequence.
- **DataRepresentationIdSeq value** () const

  Gets the ***DataRepresentation*** *(p. 1113).*
- **DataRepresentation** & **value** (const **DataRepresentationIdSeq** &source)

  Sets the data representation.
- const **rti::core::CompressionSettings** & **compression_settings** () const

  *<<**extension**>> (p. 116) Gets the compression settings by const reference (see setter).*
- **rti::core::CompressionSettings** & **compression_settings** ()

  *<<**extension**>> (p. 116) Gets the compression settings by non-const reference (see setter)*
- **dds::core::policy::DataRepresentation** & **compression_settings** (const **rti::core::CompressionSettings** &compression_settings)

  *<<**extension**>> (p. 116) Sets the compression settings.*

Static Public Member Functions

- static **DataRepresentationId xcdr** ()

  Extended Common Data Representation encoding version 1.
- static **DataRepresentationId xml** ()

  XML Data Representation (unsupported)
- static **DataRepresentationId xcdr2** ()

  Extended Common Data Representation encoding version 2.
- static **DataRepresentationId auto_id** ()

  Representation automatically chosen based on the type.
- static **DataRepresentation create_empty** () noexcept

  Unlike the default constructor, which returns a ***DataRepresentation*** *(p. 1113) instance with one element, auto_id, this static methods returns an empty* ***DataRepresentation*** *(p. 1113) instance.*

8.63.1 Detailed Description

Contains the data representations supported by entities.

Entity:

dds::topic::Topic (p. 2474), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Status:

dds::core::status::StatusMask::offered_incompatible_qos() (p. 2380), **dds::core::status::StatusMask↩ ::requested_incompatible_qos()** (p. 2381)

Properties:

RxO (p. 514) = YES
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

See also

DATA_REPRESENTATION (p. 519)

This policy has request-offer semantics for both representation and compression. For representation, a **dds::pub::↩ DataWriter** (p. 1138) may only offer a single representation. Attempting to put multiple representations in a **dds::pub↩ ::DataWriter** (p. 1138) will result in **dds::core::InconsistentPolicyError** (p. 1610). A **dds::pub::DataWriter** (p. 1138) will use its offered policy to communicate with its matched **dds::sub::DataReader** (p. 976) entities. A **dds::sub::Data↩ Reader** (p. 976) requests one or more representations. If a **dds::pub::DataWriter** (p. 1138) offers a representation that is contained within the sequence of the **dds::sub::DataReader** (p. 976), the offer satisfies the request and the policies are compatible. Otherwise, they are incompatible.

When representations are specified in the **dds::topic::qos::TopicQos** (p. 2509), **dds::pub::qos::DataWriterQos↩ ::operator=(const dds::topic::qos::TopicQos&)** (p. 1230) copies the first element of the sequence, and **dds::sub↩ ::qos::DataReaderQos::operator=(const dds::topic::qos::TopicQos&)** (p. 1076) copies the whole sequence.

For Compression, only the **rti::core::CompressionSettings::compression_ids** (p. 945) is propagated and a **dds↩ ::pub::DataWriter** (p. 1138) may only offer a single compression id. Attempting to put multiple compression_ids in a **dds::pub::DataWriter** (p. 1138) will result in **dds::core::InconsistentPolicyError** (p. 1610).

A **dds::pub::DataWriter** (p. 1138) will use its offered compression algorithm to compress data that it sends to its matched **dds::sub::DataReader** (p. 976) entities. A **dds::sub::DataReader** (p. 976) requests zero or more compression algorithms. If a **dds::pub::DataWriter** (p. 1138) offers a representation that is contained within the algorithms requested by the **dds::sub::DataReader** (p. 976), the offer satisfies the request and the policies are compatible. Otherwise, they are incompatible.

When compression IDs are specified in the **dds::topic::qos::TopicQos** (p. 2509), **dds::pub::qos::DataWriter↩ Qos::operator=(const dds::topic::qos::TopicQos&)** (p. 1230) copies a single compression ID (see **rti::core::↩ CompressionSettings::compression_ids** (p. 945)), and **dds::sub::qos::DataReaderQos::operator=(const dds↩ ::topic::qos::TopicQos&)** (p. 1076) copies all of the compression_ids.

8.63.2 Constructor & Destructor Documentation

8.63.2.1 DataRepresentation() [1/3]

```
dds::core::policy::DataRepresentation::DataRepresentation ( )  [inline]
```

Creates an instance with one element, **auto_id()** (p. 521).

8.63.2.2 DataRepresentation() [2/3]

```
dds::core::policy::DataRepresentation::DataRepresentation (
            const  DataRepresentationIdSeq & seq )  [inline], [explicit]
```

Creates an instance with a sequence of DataRepresentationId.

Parameters

seq	A vector containing the DataRepresentationIds to create this **DataRepresentation** (p. 1113)

8.63.2.3 DataRepresentation() [3/3]

```
dds::core::policy::DataRepresentation::DataRepresentation (
            const  DataRepresentationId * value_begin,
            const  DataRepresentationId * value_end )  [inline]
```

Creates an instance with a sequence of DataRepresentationId.

Parameters

value_begin	Beginning of a range of DataRepresentationId
value_end	End of the range

8.63.3 Member Function Documentation

8.63.3.1 create_empty()

```
static DataRepresentation dds::core::policy::DataRepresentation::create_empty ( ) [inline],
[static], [noexcept]
```

Unlike the default constructor, which returns a **DataRepresentation** (p. 1113) instance with one element, auto_id, this static methods returns an empty **DataRepresentation** (p. 1113) instance.

8.63.3.2 value() [1/3]

```
template<typename ShortIter >
DataRepresentation & dds::core::policy::DataRepresentation::value (
            ShortIter the_begin,
            ShortIter the_end ) [inline]
```

Sets the DataRepresentationId sequence.

Template Parameters

ShortIter	An input iterator of DataRepresentationId.

Parameters

the_begin	Beginning of the range
the_end	End of the range

8.63.3.3 value() [2/3]

```
DataRepresentationIdSeq dds::core::policy::DataRepresentation::value ( ) const [inline]
```

Gets the **DataRepresentation** (p. 1113).

Returns

the sequence of DataRepresentationId representing the data representations

8.63.3.4 value() [3/3]

```
DataRepresentation & dds::core::policy::DataRepresentation::value (
            const DataRepresentationIdSeq & source ) [inline]
```

Sets the data representation.

Parameters

source	The vector where the DataRepresentationId will be copied from

8.63.3.5 compression_settings() [1/3]

```
const  rti::core::CompressionSettings & compression_settings ( ) const
```

<<***extension***>> *(*p. *116)* Gets the compression settings by const reference (see setter).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.63.3.6 compression_settings() [2/3]

```
rti::core::CompressionSettings & compression_settings ( )
```

<<***extension***>> *(*p. *116)* Gets the compression settings by non-const reference (see setter)

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.63.3.7 compression_settings() [3/3]

```
dds::core::policy::DataRepresentation & compression_settings (
            const  rti::core::CompressionSettings & compression_settings )
```

<<***extension***>> *(*p. *116)* Sets the compression settings.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

compression_ids:
[default] The value depends on the entity
dds::pub::DataWriter (p. 1138) and **dds::topic::Topic** (p. 2474) are set to **rti::core::CompressionIdMask::none()** (p. 942)
dds::sub::DataReader (p. 976) **rti::core::CompressionIdMask::all()** (p. 941)

writer_compression_level:
[default] The value depends on the entity
dds::pub::DataWriter (p. 1138) and **dds::topic::Topic** (p. 2474) are set to **rti::core::CompressionSettings**↩ **::compression_level_default()** (p. 945)
dds::sub::DataReader (p. 976) NOT SUPPORTED

writer_compression_threshold:
[default] The value depends on the entity
dds::pub::DataWriter (p. 1138) and **dds::topic::Topic** (p. 2474) are set to rti::core::CompressionSettings↩ ::compression_threshold_default()
dds::sub::DataReader (p. 976) NOT SUPPORTED

See also

rti::core::CompressionSettings (p. 943)

Categories

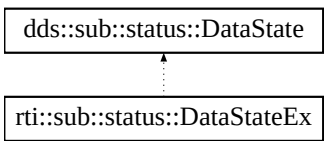
Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Sending Large Data** (p. 197)

8.64 dds::sub::status::DataState Class Reference

The **DataState** (p. 1118) class describes the state of a sample and includes the information about the sample's **InstanceState** (p. 1615), **ViewState** (p. 2614), and **SampleState** (p. 2316).

```
#include <dds/sub/status/DataState.hpp>
```

Inheritance diagram for dds::sub::status::DataState:



Public Member Functions

- **DataState** ()

  *Create a **DataState** (p. 1118) with **InstanceState::any()** (p. 1618), **ViewState::any()** (p. 2616), and **SampleState::any()** (p. 2318)*
- **DataState** (const **SampleState** &the_sample_state)

  *Create a **DataState** (p. 1118) with **InstanceState::any()** (p. 1618), **ViewState::any()** (p. 2616), and the provided **SampleState** (p. 2316).*
- **DataState** (const **ViewState** &the_view_state)

  *Create a **DataState** (p. 1118) with **InstanceState::any()** (p. 1618), **SampleState::any()** (p. 2318), and the provided **ViewState** (p. 2614).*
- **DataState** (const **InstanceState** &the_instance_state)

  *Create a **DataState** (p. 1118) with **SampleState::any()** (p. 2318), **ViewState::any()** (p. 2616), and the provided **InstanceState** (p. 1615).*
- **DataState** (const **SampleState** &the_sample_state, const **ViewState** &the_view_state, const **InstanceState** &the_instance_state)

  *Create a **DataState** (p. 1118) with the provided **SampleState** (p. 2316), **ViewState** (p. 2614), and **InstanceState** (p. 1615).*
- bool **operator==** (const **DataState** &other) const

  *Compare two **DataState** (p. 1118) objects for equality.*
- bool **operator!=** (const **DataState** &other) const

  *Compare two **DataState** (p. 1118) objects for inequality.*
- **DataState** & **operator<<** (const **SampleState** &the_sample_state)

  *Set the provided **SampleState** (p. 2316) in a **DataState** (p. 1118) object.*
- **DataState** & **operator<<** (const **InstanceState** &the_instance_state)

  *Set the provided **InstanceState** (p. 1615) in a **DataState** (p. 1118) object.*
- **DataState** & **operator<<** (const **ViewState** &the_view_state)

  *Set the provided **ViewState** (p. 2614) in a **DataState** (p. 1118) object.*
- const **DataState** & **operator>>** (**SampleState** &the_sample_state) const

  *Get the **SampleState** (p. 2316) from a **DataState** (p. 1118) object.*
- const **DataState** & **operator>>** (**InstanceState** &the_instance_state) const

  *Get the **InstanceState** (p. 1615) from a **DataState** (p. 1118) object.*
- const **DataState** & **operator>>** (**ViewState** &the_view_state) const

  *Get the **ViewState** (p. 2614) from a **DataState** (p. 1118) object.*
- const **SampleState** & **sample_state** () const

  *Get the **SampleState** (p. 2316) from a **DataState** (p. 1118) object.*
- void **sample_state** (const **SampleState** &the_sample_state)

  *Set the provided **SampleState** (p. 2316) in a **DataState** (p. 1118) object.*
- const **InstanceState** & **instance_state** () const

  *Get the **InstanceState** (p. 1615) from a **DataState** (p. 1118) object.*
- void **instance_state** (const **InstanceState** &the_instance_state)

  *Set the provided **InstanceState** (p. 1615) in a **DataState** (p. 1118) object.*
- const **ViewState** & **view_state** () const

  *Get the **ViewState** (p. 2614) from a **DataState** (p. 1118) object.*
- void **view_state** (const **ViewState** &the_view_state)

  *Set the provided **ViewState** (p. 2614) in a **DataState** (p. 1118) object.*

Static Public Member Functions

- static **DataState** **any** ()

  *Create a **DataState** (p. 1118) with **InstanceState::any()** (p. 1618), **ViewState::any()** (p. 2616), and **SampleState::any()** (p. 2318)*
- static **DataState** **new_data** ()

  *Create a **DataState** (p. 1118) with **InstanceState::alive()** (p. 1617), **ViewState::any()** (p. 2616), and **SampleState::not_read()** (p. 2318)*
- static **DataState** **any_data** ()

  *Create a **DataState** (p. 1118) with **InstanceState::alive()** (p. 1617), **ViewState::any()** (p. 2616), and **SampleState::any()** (p. 2318)*
- static **DataState** **new_instance** ()

  *Create a **DataState** (p. 1118) with **InstanceState::alive()** (p. 1617), **ViewState::new_view()** (p. 2616), and **SampleState::any()** (p. 2318)*

Friends

- void **swap** (**DataState** &left, **DataState** &right) OMG_NOEXCEPT

  *Swap the contents of two **DataState** (p. 1118) objects.*

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator<<** (std::ostream &os, const **DataState** &s)

  Prints a data state as a readable string.

8.64.1 Detailed Description

The **DataState** (p. 1118) class describes the state of a sample and includes the information about the sample's **InstanceState** (p. 1615), **ViewState** (p. 2614), and **SampleState** (p. 2316).

8.64.2 Constructor & Destructor Documentation

8.64.2.1 DataState() [1/5]

```
dds::sub::status::DataState::DataState ( )  [inline]
```

Create a **DataState** (p. 1118) with **InstanceState::any()** (p. 1618), **ViewState::any()** (p. 2616), and **SampleState::any()** (p. 2318)

Referenced by **any()**, **any_data()**, **new_data()**, and **new_instance()**.

8.64.2.2 DataState() [2/5]

```
dds::sub::status::DataState::DataState (
            const  SampleState & the_sample_state )  [inline]
```

Create a **DataState** (p. 1118) with **InstanceState::any()** (p. 1618), **ViewState::any()** (p. 2616), and the provided **SampleState** (p. 2316).

Note that the constructor is implicit, so you can use a **SampleState** (p. 2316) wherever a **DataState** (p. 1118) is expected.

8.64.2.3 DataState() [3/5]

```
dds::sub::status::DataState::DataState (
            const  ViewState & the_view_state )  [inline]
```

Create a **DataState** (p. 1118) with **InstanceState::any()** (p. 1618), **SampleState::any()** (p. 2318), and the provided **ViewState** (p. 2614).

Note that the constructor is implicit, so you can use a **ViewState** (p. 2614) wherever a **DataState** (p. 1118) is expected.

8.64.2.4 DataState() [4/5]

```
dds::sub::status::DataState::DataState (
            const  InstanceState & the_instance_state )  [inline]
```

Create a **DataState** (p. 1118) with **SampleState::any()** (p. 2318), **ViewState::any()** (p. 2616), and the provided **InstanceState** (p. 1615).

Note that the constructor is implicit, so you can use a **InstanceState** (p. 1615) wherever a **DataState** (p. 1118) is expected.

8.64.2.5 DataState() [5/5]

```
dds::sub::status::DataState::DataState (
            const  SampleState & the_sample_state,
            const  ViewState & the_view_state,
            const  InstanceState & the_instance_state )  [inline]
```

Create a **DataState** (p. 1118) with the provided **SampleState** (p. 2316), **ViewState** (p. 2614), and **InstanceState** (p. 1615).

8.64.3 Member Function Documentation

8.64.3.1 operator==()

```
bool dds::sub::status::DataState::operator== (
            const  DataState & other ) const  [inline]
```

Compare two **DataState** (p. 1118) objects for equality.

References **instance_state()**, **sample_state()**, and **view_state()**.

Referenced by **operator!=()**.

8.64.3.2 operator"!=()

```
bool dds::sub::status::DataState::operator!= (
            const  DataState & other ) const  [inline]
```

Compare two **DataState** (p. 1118) objects for inequality.

References **operator==()**.

8.64.3.3 operator<<() [1/3]

```
DataState & dds::sub::status::DataState::operator<< (
            const  SampleState & the_sample_state )  [inline]
```

Set the provided **SampleState** (p. 2316) in a **DataState** (p. 1118) object.

8.64.3.4 operator<<() [2/3]

```
DataState & dds::sub::status::DataState::operator<< (
            const  InstanceState & the_instance_state )  [inline]
```

Set the provided **InstanceState** (p. 1615) in a **DataState** (p. 1118) object.

8.64.3.5 operator<<() [3/3]

```
DataState & dds::sub::status::DataState::operator<< (
            const  ViewState & the_view_state )  [inline]
```

Set the provided **ViewState** (p. 2614) in a **DataState** (p. 1118) object.

8.64.3.6 operator>>() [1/3]

```
const  DataState & dds::sub::status::DataState::operator>> (
             SampleState & the_sample_state ) const  [inline]
```

Get the **SampleState** (p. 2316) from a **DataState** (p. 1118) object.

8.64.3.7 operator>>() [2/3]

```
const  DataState & dds::sub::status::DataState::operator>> (
             InstanceState & the_instance_state ) const  [inline]
```

Get the **InstanceState** (p. 1615) from a **DataState** (p. 1118) object.

8.64.3.8 operator>>() [3/3]

```
const  DataState & dds::sub::status::DataState::operator>> (
             ViewState & the_view_state ) const  [inline]
```

Get the **ViewState** (p. 2614) from a **DataState** (p. 1118) object.

8.64.3.9 sample_state() [1/2]

```
const  SampleState & dds::sub::status::DataState::sample_state ( ) const  [inline]
```

Get the **SampleState** (p. 2316) from a **DataState** (p. 1118) object.

Referenced by **dds::sub::find()**, and **operator==()**.

8.64.3.10 sample_state() [2/2]

```
void dds::sub::status::DataState::sample_state (
            const  SampleState & the_sample_state )  [inline]
```

Set the provided **SampleState** (p. 2316) in a **DataState** (p. 1118) object.

8.64.3.11 instance_state() [1/2]

```
const InstanceState & dds::sub::status::DataState::instance_state ( ) const [inline]
```

Get the **InstanceState** (p. 1615) from a **DataState** (p. 1118) object.

Referenced by **dds::sub::find()**, and **operator==()**.

8.64.3.12 instance_state() [2/2]

```
void dds::sub::status::DataState::instance_state (
            const InstanceState & the_instance_state ) [inline]
```

Set the provided **InstanceState** (p. 1615) in a **DataState** (p. 1118) object.

8.64.3.13 view_state() [1/2]

```
const ViewState & dds::sub::status::DataState::view_state ( ) const [inline]
```

Get the **ViewState** (p. 2614) from a **DataState** (p. 1118) object.

Referenced by **dds::sub::find()**, and **operator==()**.

8.64.3.14 view_state() [2/2]

```
void dds::sub::status::DataState::view_state (
            const ViewState & the_view_state ) [inline]
```

Set the provided **ViewState** (p. 2614) in a **DataState** (p. 1118) object.

8.64.3.15 any()

```
static DataState dds::sub::status::DataState::any ( ) [inline], [static]
```

Create a **DataState** (p. 1118) with **InstanceState::any()** (p. 1618), **ViewState::any()** (p. 2616), and **SampleState::any()** (p. 2318)

Examples

Foo_subscriber.cxx.

References **dds::sub::status::SampleState::any()**, **dds::sub::status::ViewState::any()**, **dds::sub::status::↩ InstanceState::any()**, and **DataState()**.

8.64.3.16 new_data()

```
static DataState dds::sub::status::DataState::new_data ( ) [inline], [static]
```

Create a **DataState** (p. 1118) with **InstanceState::alive()** (p. 1617), **ViewState::any()** (p. 2616), and **SampleState::not_read()** (p. 2318)

References **dds::sub::status::InstanceState::alive()**, **dds::sub::status::ViewState::any()**, **DataState()**, and **dds::sub::status::SampleState::not_read()**.

8.64.3.17 any_data()

```
static DataState dds::sub::status::DataState::any_data ( ) [inline], [static]
```

Create a **DataState** (p. 1118) with **InstanceState::alive()** (p. 1617), **ViewState::any()** (p. 2616), and **SampleState::any()** (p. 2318)

References **dds::sub::status::InstanceState::alive()**, **dds::sub::status::SampleState::any()**, **dds::sub::status::ViewState::any()**, and **DataState()**.

8.64.3.18 new_instance()

```
static DataState dds::sub::status::DataState::new_instance ( ) [inline], [static]
```

Create a **DataState** (p. 1118) with **InstanceState::alive()** (p. 1617), **ViewState::new_view()** (p. 2616), and **SampleState::any()** (p. 2318)

References **dds::sub::status::InstanceState::alive()**, **dds::sub::status::SampleState::any()**, **DataState()**, and **dds::sub::status::ViewState::new_view()**.

8.64.4 Friends And Related Function Documentation

8.64.4.1 swap

```
void swap (
            DataState & left,
            DataState & right ) [friend]
```

Swap the contents of two **DataState** (p. 1118) objects.

8.64.4.2 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const  DataState & s )  [related]
```

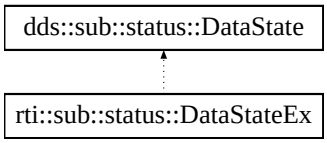
Prints a data state as a readable string.

8.65 rti::sub::status::DataStateEx Class Reference

<<***extension***>> (p. *354)* <<***value-type***>> (p. *350)* An extended version of **dds::sub::status::DataState** (p. 1118) that also contains **StreamKind** (p. 2390)

```
#include <DataStateEx.hpp>
```

Inheritance diagram for rti::sub::status::DataStateEx:



Public Member Functions

- **DataStateEx** ()

  *Create a **DataStateEx** (p. 1126) with InstanceState::any(), ViewState::any(), SampleState::any(), and **StreamKind::any()** (p. 2392).*
- **DataStateEx** (const **StreamKind** &the_stream_kind)

  *Create a **DataStateEx** (p. 1126) with InstanceState::any(), ViewState::any(), SampleState::any() and the provided **StreamKind** (p. 2390).*
- **DataStateEx** (const **dds::sub::status::DataState** &the_data_state)

  *Create a **DataStateEx** (p. 1126) with the provided SampleState, ViewState, InstanceState, and **StreamKind::any()** (p. 2392)*
- **DataStateEx** (const **dds::sub::status::DataState** &the_data_state, const **StreamKind** &the_stream_kind)

  *Create a **DataStateEx** (p. 1126) with the provided SampleState, ViewState, InstanceState, and **StreamKind** (p. 2390).*
- **DataStateEx** & **operator**<< (const **StreamKind** &the_stream_kind)

  *Set the **StreamKind** (p. 2390).*
- const **DataStateEx** & **operator**>> (**StreamKind** &the_stream_kind) const

  *Get the **StreamKind** (p. 2390).*
- const **StreamKind** & **stream_kind** () const

  *Get the **StreamKind** (p. 2390).*
- void **stream_kind** (const **StreamKind** &the_stream_kind)

  *Set the **StreamKind** (p. 2390).*
- const **DataState** & **data_state** () const

  Access the view, sample and instance states.

- bool **operator==** (const **DataStateEx** &other) const

  *Compare two **DataStateEx** (p. 1126) objects for equality.*
- bool **operator!=** (const **DataStateEx** &other) const

  *Compare two **DataStateEx** (p. 1126) objects for inequality.*
- const SampleState & **sample_state** () const

  Access the SampleState.
- void **sample_state** (const SampleState &the_sample_state)

  Access the SampleState.
- const ViewState & **view_state** () const

  Access the ViewState.
- void **view_state** (const ViewState &the_view_state)

  Access the ViewState.
- const InstanceState & **instance_state** () const

  Access the InstanceState.
- void **instance_state** (const InstanceState &the_instance_state)

  Access the InstanceState.

Static Public Member Functions

- static **DataState** **any** ()

  Create a DataState with InstanceState::any(), ViewState::any(), and SampleState::any()
- static **DataState** **new_data** ()

  Create a DataState with InstanceState::alive(), ViewState::any(), and SampleState::not_read()
- static **DataState** **any_data** ()

  Create a DataState with InstanceState::alive(), ViewState::any(), and SampleState::any()
- static **DataState** **new_instance** ()

  Create a DataState with InstanceState::alive(), ViewState::new_view(), and SampleState::any()

Friends

- void **swap** (**DataStateEx** &left, **DataStateEx** &right) OMG_NOEXCEPT

  Swap the contents of two DataState objects.

Additional Inherited Members

8.65.1 Detailed Description

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* An extended version of **dds::sub::status::DataState** (p. 1118) that also contains **StreamKind** (p. 2390)

DataStateEx (p. 1126) is an extension of DataState that allows setting an additional state mask: the **StreamKind** (p. 2390). It is used by rti::sub::cond::create_read_condition_ex and rti::sub::cond::create_query_condition_ex to create conditions that can filter based on the **StreamKind** (p. 2390).

A **DataStateEx** (p. 1126) contains the state masks InstanceState, ViewState and SampleState (with the same getters and setters: **instance_state()** (p. 1131), **view_state()** (p. 1130) and **sample_state()** (p. 1130)).

In addition it contains the mask **StreamKind** (p. 2390) with its getter and setter: **stream_kind()** (p. 1129).

8.65.2 Constructor & Destructor Documentation

8.65.2.1 DataStateEx() [1/4]

```
rti::sub::status::DataStateEx::DataStateEx ( )  [inline]
```

Create a **DataStateEx** (p. 1126) with InstanceState::any(), ViewState::any(), SampleState::any(), and **StreamKind↩ ::any()** (p. 2392).

8.65.2.2 DataStateEx() [2/4]

```
rti::sub::status::DataStateEx::DataStateEx (
            const  StreamKind & the_stream_kind )  [inline]
```

Create a **DataStateEx** (p. 1126) with InstanceState::any(), ViewState::any(), SampleState::any() and the provided **StreamKind** (p. 2390).

Note that the constructor is implicit, so you can use a **StreamKind** (p. 2390) wherever a **DataStateEx** (p. 1126) is expected.

8.65.2.3 DataStateEx() [3/4]

```
rti::sub::status::DataStateEx::DataStateEx (
            const  dds::sub::status::DataState & the_data_state )  [inline]
```

Create a **DataStateEx** (p. 1126) with the provided SampleState, ViewState, InstanceState, and **StreamKind::any()** (p. 2392)

The constructor is implicit, so DataState can automatically be converted to a **DataStateEx** (p. 1126) with **StreamKind↩ ::any()** (p. 2392).

Parameters

the_data_state	Contains the SampleState, ViewState and InstanceState

8.65.2.4 DataStateEx() [4/4]

```
rti::sub::status::DataStateEx::DataStateEx (
            const  dds::sub::status::DataState & the_data_state,
            const  StreamKind & the_stream_kind )  [inline]
```

Create a **DataStateEx** (p. 1126) with the provided SampleState, ViewState, InstanceState, and **StreamKind** (p. 2390).

Parameters

the_data_state	Contains the SampleState, ViewState and InstanceState
the_stream_kind	Contains the **StreamKind** (p. 2390)

8.65.3 Member Function Documentation

8.65.3.1 operator<<()

```
DataStateEx & rti::sub::status::DataStateEx::operator<< (
            const  StreamKind & the_stream_kind )  [inline]
```

Set the **StreamKind** (p. 2390).

8.65.3.2 operator>>()

```
const  DataStateEx & rti::sub::status::DataStateEx::operator>> (
             StreamKind & the_stream_kind ) const  [inline]
```

Get the **StreamKind** (p. 2390).

8.65.3.3 stream_kind() [1/2]

```
const  StreamKind & rti::sub::status::DataStateEx::stream_kind ( ) const  [inline]
```

Get the **StreamKind** (p. 2390).

Referenced by **operator==()**.

8.65.3.4 stream_kind() [2/2]

```
void rti::sub::status::DataStateEx::stream_kind (
            const  StreamKind & the_stream_kind )  [inline]
```

Set the **StreamKind** (p. 2390).

8.65.3.5 data_state()

```
const  DataState & rti::sub::status::DataStateEx::data_state ( ) const  [inline]
```

Access the view, sample and instance states.

Referenced by **operator==()**.

8.65.3.6 operator==()

```
bool rti::sub::status::DataStateEx::operator== (
            const  DataStateEx & other ) const  [inline]
```

Compare two **DataStateEx** (p. 1126) objects for equality.

References **data_state()**, and **stream_kind()**.

8.65.3.7 operator"!=()

```
bool rti::sub::status::DataStateEx::operator!= (
            const  DataStateEx & other ) const  [inline]
```

Compare two **DataStateEx** (p. 1126) objects for inequality.

8.65.3.8 sample_state() [1/2]

```
const SampleState & dds::sub::status::DataState::sample_state ( ) const  [inline]
```

Access the SampleState.

8.65.3.9 sample_state() [2/2]

```
void dds::sub::status::DataState::sample_state (
            const SampleState & the_sample_state )  [inline]
```

Access the SampleState.

8.65.3.10 view_state() [1/2]

```
const ViewState & dds::sub::status::DataState::view_state ( ) const  [inline]
```

Access the ViewState.

8.65.3.11 view_state() [2/2]

```
void dds::sub::status::DataState::view_state (
            const ViewState & the_view_state )  [inline]
```

Access the ViewState.

8.65.3.12 instance_state() [1/2]

```
const InstanceState & dds::sub::status::DataState::instance_state ( ) const  [inline]
```

Access the InstanceState.

8.65.3.13 instance_state() [2/2]

```
void dds::sub::status::DataState::instance_state (
            const InstanceState & the_instance_state )  [inline]
```

Access the InstanceState.

8.65.3.14 any()

```
static  DataState dds::sub::status::DataState::any ( )  [inline], [static]
```

Create a DataState with InstanceState::any(), ViewState::any(), and SampleState::any()

8.65.3.15 new_data()

```
static DataState dds::sub::status::DataState::new_data ( ) [inline], [static]
```

Create a DataState with InstanceState::alive(), ViewState::any(), and SampleState::not_read()

8.65.3.16 any_data()

```
static DataState dds::sub::status::DataState::any_data ( ) [inline], [static]
```

Create a DataState with InstanceState::alive(), ViewState::any(), and SampleState::any()

8.65.3.17 new_instance()

```
static DataState dds::sub::status::DataState::new_instance ( ) [inline], [static]
```

Create a DataState with InstanceState::alive(), ViewState::new_view(), and SampleState::any()

8.65.4 Friends And Related Function Documentation

8.65.4.1 swap

```
void swap (
            DataStateEx & left,
            DataStateEx & right ) [friend]
```

Swap the contents of two DataState objects.

8.66 dds::core::policy::DataTag Class Reference

Stores name-value (string) pairs that can be used to determine access permissions.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Types

- typedef std::pair< std::string, std::string > **Entry**

  *The type of the elements that **DataTag** (p. 1132) contains.*

Public Member Functions

- **DataTag** ()

  Creates a policy with an empty sequence of tags.
- template<typename EntryIter >
  DataTag (EntryIter begin, EntryIter end)

  *Creates a **DataTag** (p. 1132) container with the entries specified by an iterator range.*
- **DataTag** (std::initializer_list< **Entry** > entries)

  <<***C++11***>> *(p. 353) Creates a **DataTag** (p. 1132) container with entries from an initializer_list*
- void **set** (std::initializer_list< **Entry** > entries)

  <<***C++11***>> *(p. 353) Adds tags from an initializer_list*
- bool **exists** (const std::string &key) const

  Returns true if a tag exists.
- std::string **get** (const std::string &key) const

  Returns the value of a tag identified by a key if it exists.
- **rti::core::optional_value**< std::string > **try_get** (const std::string &key) const

  Returns the value of a tag identified by a key or an empty optional_value if it doesn't exist.
- **DataTag** & **set** (const **Entry** &tag)

  Adds or assigns a tag from a pair of strings.
- template<typename EntryIter >
  DataTag & **set** (EntryIter begin, EntryIter end)

  Adds a range of tags.
- bool **remove** (const std::string &key)

  Removes the tag identified by a key.
- size_t **size** () const

  Returns the number of tags.
- std::map< std::string, std::string > **get_all** () const

  Retrieves a copy of all the entries in a std::map.

8.66.1 Detailed Description

Stores name-value (string) pairs that can be used to determine access permissions.

Entity:

dds::sub::DataReader (p. 976) **dds::pub::DataWriter** (p. 1138)

Properties:

RxO (p. 514) = N/A;
Changeable (p. 514) = **NO** (p. 514)

8.66.2 Usage

The DATA_TAG QoS policy can be used to associate a set of tags in the form of (name, value) pairs with a **dds::sub↩ ::DataReader** (p. 976) or **dds::pub::DataWriter** (p. 1138). This is similar to the **rti::core::policy::Property** (p. 1969), except for the following differences:

- Data tags are always propagated. You cannot select whether or not a particular pair should be propagated.
- Data tags are not exposed to API functions, such as **dds::pub::matched_subscription_data()** (p. 650), that receive **dds::topic::PublicationBuiltinTopicData** (p. 1977) or **dds::topic::SubscriptionBuiltinTopic↩ Data** (p. 2428) as a parameter.
- Connext passes data tags to the Access Control Security Plugin, which may use them to decide whether to allow or deny the corresponding entities.

There are helper functions to facilitate working with data tags. See the **DATA_TAG** (p. 523) page.

8.66.3 Member Typedef Documentation

8.66.3.1 Entry

```
typedef std::pair<std::string, std::string>  dds::core::policy::DataTag::Entry
```

The type of the elements that **DataTag** (p. 1132) contains.

Key/value string pairs.

8.66.4 Constructor & Destructor Documentation

8.66.4.1 DataTag() [1/3]

```
dds::core::policy::DataTag::DataTag ( )  [inline]
```

Creates a policy with an empty sequence of tags.

8.66.4.2 DataTag() [2/3]

```
template<typename EntryIter >
dds::core::policy::DataTag::DataTag (
            EntryIter begin,
            EntryIter end )  [inline]
```

Creates a **DataTag** (p. 1132) container with the entries specified by an iterator range.

See also

set(EntryIter, EntryIter) (p. 1136)

8.66.4.3 DataTag() [3/3]

```
dds::core::policy::DataTag::DataTag (
            std::initializer_list<  Entry > entries )  [inline]
```

<<***C++11***>> *(p. 353)* Creates a **DataTag** (p. 1132) container with entries from an initializer_list

For example:
```
DataTag my_tags {{"key 1", "value 1"}, {"key 2", "value 2"}};
```

If a key is duplicated, only one entry will be inserted, with the value that comes last.

8.66.5 Member Function Documentation

8.66.5.1 set() [1/3]

```
void dds::core::policy::DataTag::set (
            std::initializer_list<  Entry > entries )  [inline]
```

<<***C++11***>> *(p. 353)* Adds tags from an initializer_list

8.66.5.2 exists()

```
bool dds::core::policy::DataTag::exists (
            const std::string & key ) const  [inline]
```

Returns true if a tag exists.

8.66.5.3 get()

```
std::string dds::core::policy::DataTag::get (
            const std::string & key ) const  [inline]
```

Returns the value of a tag identified by a key if it exists.

If the tag doesn't exist it throws **dds::core::PreconditionNotMetError** (p. 1941).

8.66.5.4 try_get()

```
rti::core::optional_value< std::string > dds::core::policy::DataTag::try_get (
            const std::string & key ) const  [inline]
```

Returns the value of a tag identified by a key or an empty optional_value if it doesn't exist.

8.66.5.5 set() [2/3]

```
DataTag & dds::core::policy::DataTag::set (
            const  Entry & tag )  [inline]
```

Adds or assigns a tag from a pair of strings.

If the key doesn't exist it adds the new entry. Otherwise it overrides its value with the new one.

Parameters

tag	`tag.first` is the key and `tag.second` is the value

8.66.5.6 set() [3/3]

```
template<typename EntryIter >
DataTag & dds::core::policy::DataTag::set (
            EntryIter begin,
            EntryIter end )  [inline]
```

Adds a range of tags.

If a key is duplicated, only one entry will remain, with the value that comes last.

Template Parameters

EntryIter	A forward iterator whose value type is **DataTag::Entry** (p. 1134), such as the iterators of a `std::map<std::string, std::string>`.

Parameters

begin	Beginning of a range of Entry
end	End of that range

8.66.5.7 remove()

```
bool dds::core::policy::DataTag::remove (
             const std::string & key )  [inline]
```

Removes the tag identified by a key.

Returns

true if the tag was removed or false if it didn't exist.

8.66.5.8 size()

```
size_t dds::core::policy::DataTag::size ( ) const  [inline]
```

Returns the number of tags.

8.66.5.9 get_all()

```
std::map< std::string, std::string > dds::core::policy::DataTag::get_all ( ) const  [inline]
```

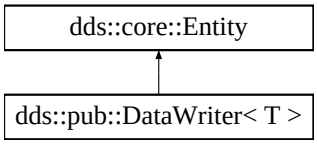
Retrieves a copy of all the entries in a std::map.

8.67 dds::pub::DataWriter< T > Class Template Reference

<<***reference-type***>> *(p. 351)* Allows an application to publish data for a **dds::topic::Topic** (p. 2474)

`#include <dds/pub/DataWriter.hpp>`

Inheritance diagram for dds::pub::DataWriter< T >:



Public Member Functions

- **DataWriter** (const **dds::pub::Publisher** &pub, const **dds::topic::Topic**< T > &the_topic)

  *Creates a **DataWriter** (p. 1138).*
- **DataWriter** (const **dds::pub::Publisher** &pub, const **dds::topic::Topic**< T > &the_topic, const **dds::pub::qos::DataWriterQos** &the_qos, **dds::pub::DataWriterListener**< T > *the_listener=NULL, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Creates a **DataWriter** (p. 1138) with QoS and listener.*
- **DataWriter** (const **dds::pub::Publisher** &pub, const **dds::topic::Topic**< T > &the_topic, const **dds::pub::qos::DataWriterQos** &the_qos, std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Creates a **DataWriter** (p. 1138) with QoS and listener.*
- **DataWriter** (const **dds::topic::Topic**< T > & **topic**)

  *Creates a **DataWriter** (p. 1138).*
- **DataWriter** (const **dds::topic::Topic**< T > & **topic**, const **dds::pub::qos::DataWriterQos** & **qos**)

  *Creates a **DataWriter** (p. 1138) with QoS.*
- **DataWriter** (const **dds::topic::Topic**< T > & **topic**, const **dds::pub::qos::DataWriterQos** & **qos**, std::shared_ptr< **Listener** > **listener**, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Creates a **DataWriter** (p. 1138) with QoS and listener.*
- void **write** (const T &instance_data)

  Modifies the value of a data instance.
- void **write** (const T &instance_data, const **dds::core::Time** &timestamp)

  Modifies the value of a data instance and specifies the timestamp.
- void **write** (const T &instance_data, const **dds::core::InstanceHandle** &handle)

  Modifies the value of a data instance.
- void **write** (const T &instance_data, const **dds::core::InstanceHandle** &handle, const **dds::core::Time** &source_timestamp)

  Modifies the value of a data instance and specifies the timestamp.
- void **write** (const **dds::topic::TopicInstance**< T > &topic_instance)

  *Write a **dds::topic::TopicInstance** (p. 2505).*
- void **write** (const **dds::topic::TopicInstance**< T > &topic_instance, const **dds::core::Time** &timestamp)

  Write a topic instance with time stamp.

- template<typename FWIterator >
  void **write** (const FWIterator &begin, const FWIterator &end)
- template<typename FWIterator >
  void **write** (const FWIterator &begin, const FWIterator &end, const **dds::core::Time** &timestamp)
- template<typename SamplesFWIterator , typename HandlesFWIterator >
  void **write** (const SamplesFWIterator &data_begin, const SamplesFWIterator &data_end, const Handles↩
  FWIterator &handle_begin, const HandlesFWIterator &handle_end)

  Write a series of samples and their parallel instance handles.
- template<typename SamplesFWIterator , typename HandlesFWIterator >
  void **write** (const SamplesFWIterator &data_begin, const SamplesFWIterator &data_end, const Handles↩
  FWIterator &handle_begin, const HandlesFWIterator &handle_end, const **dds::core::Time** &timestamp)

  Write a series of samples and their parallel instance handles and a timestamp.
- **DataWriter** & **operator**<< (const T &data)

  Writes a sample.
- **DataWriter** & **operator**<< (const std::pair< T, **dds::core::Time** > &data)

  Writes a sample with a timestamp.
- **DataWriter** & **operator**<< (const std::pair< T, **dds::core::InstanceHandle** > &data)

  Writes a sample with an instance handle.
- const **dds::core::InstanceHandle** **register_instance** (const T &instance_data)

  Informs RTI Connext that the application will be modifying a particular instance.
- const **dds::core::InstanceHandle** **register_instance** (const T &instance_data, const **dds::core::Time** &source_timestamp)

  Informs RTI Connext that the application will be modifying a particular instance and specifies the timestamp.
- **DataWriter** & **unregister_instance** (const **dds::core::InstanceHandle** &handle)

  Unregister an instance.
- **DataWriter** & **unregister_instance** (const **dds::core::InstanceHandle** &handle, const **dds::core::Time** &source_timestamp)

  Unregister an instance with timestamp.
- **DataWriter** & **dispose_instance** (const **dds::core::InstanceHandle** &handle)

  Requests the middleware to delete the instance identified by the instance handle.
- **DataWriter** & **dispose_instance** (const **dds::core::InstanceHandle** &the_instance_handle, const **dds**↩
  ::core::Time &source_timestamp)

  Dispose an instance with a timestamp.
- T & **key_value** (T &key_holder, const **dds::core::InstanceHandle** &handle)

  Retrieve the instance key that corresponds to an instance handle.
- **dds::topic::TopicInstance**< T > & **key_value** (**dds::topic::TopicInstance**< T > &key_holder, const **dds**↩
  ::core::InstanceHandle &handle)

  Retrieve the instance key that corresponds to an instance handle.
- **dds::core::InstanceHandle** **lookup_instance** (const T &key_holder)

  Retrieve the instance handle that corresponds to an instance key_holder.
- const **dds::pub::qos::DataWriterQos** **qos** () const

  Gets the DataWriterQos.
- void **qos** (const **dds::pub::qos::DataWriterQos** &the_qos)

  Sets the DataWriterQos.
- **DataWriter** & **operator**<< (const **dds::pub::qos::DataWriterQos** &the_qos)

  Set the DataWriterQos.
- const **DataWriter** & **operator**>> (**dds::pub::qos::DataWriterQos** &the_qos) const

  Get the DataWriterQos.

- const **dds::topic::Topic**< T > & **topic** () const

  *Get the Topic associated with this **DataWriter*** (p. *1138).*
- const **dds::pub::Publisher** & **publisher** () const

  *Get the **Publisher*** (p. *1993) that owns this **DataWriter*** (p. *1138).*
- void **wait_for_acknowledgments** (const **dds::core::Duration** &max_wait)

  *Blocks the calling thread until all data written by reliable **DataWriter*** (p. *1138) entity is acknowledged, or until timeout expires.*
- void **listener** (**DataWriterListener**< T > *l, const **dds::core::status::StatusMask** &mask)

  *Sets the **DataWriter*** (p. *1138) listener.*
- **DataWriterListener**< T > * **listener** () const

  *Returns the listener currently associated with this **DataWriter*** (p. *1138).*
- void **set_listener** (std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &event_↩ mask)

  Sets the listener associated with this writer.
- void **set_listener** (std::shared_ptr< **Listener** > the_listener)

  Sets the listener associated with this writer.
- std::shared_ptr< **Listener** > **get_listener** () const

  Returns the listener currently associated with this writer.
- const **dds::core::status::LivelinessLostStatus** **liveliness_lost_status** ()

  Get the LivelinessLostStatus.
- const **dds::core::status::OfferedDeadlineMissedStatus** **offered_deadline_missed_status** ()

  Get the OfferedDeadlineMissedStatus.
- const **dds::core::status::OfferedIncompatibleQosStatus** **offered_incompatible_qos_status** ()

  Get the OfferedIncompatibleQosStatus.
- const **dds::core::status::PublicationMatchedStatus** **publication_matched_status** ()

  Get the PublicationMatchedStatus.
- void **assert_liveliness** ()

  *Manually asserts the liveliness of this **DataWriter*** (p. *1138).*
- void **unregister_instance** (**rti::pub::WriteParams** &params)

  *<<**extension**>> (p. 354) Unregister an instance with parameters*
- void **dispose_instance** (**rti::pub::WriteParams** &params)

  *<<**extension**>> (p. 354) Dispose an instance with parameters*
- bool **is_sample_app_acknowledged** (const **rti::core::SampleIdentity** &sample_id)

  *<<**extension**>> (p. 354) Indicates if a sample is considered application-acknowledged*
- void **wait_for_asynchronous_publishing** (const **dds::core::Duration** &max_wait)

  *<<**extension**>> (p. 354) Blocks the calling thread until asynchronous sending is complete.*
- **rti::core::status::ReliableWriterCacheChangedStatus** **reliable_writer_cache_changed_status** ()

  *<<**extension**>> (p. 354) Get the reliable cache status for this writer.*
- **rti::core::status::ReliableReaderActivityChangedStatus** **reliable_reader_activity_changed_status** ()

  *<<**extension**>> (p. 354) Get the reliable reader activity changed status for this writer*
- **rti::core::status::DataWriterCacheStatus** **datawriter_cache_status** ()

  *<<**extension**>> (p. 354) Get the cache status for this writer*
- **rti::core::status::DataWriterProtocolStatus** **datawriter_protocol_status** ()

  *<<**extension**>> (p. 354) Get the protocol status for this writer*
- **rti::core::status::DataWriterProtocolStatus** **matched_subscription_datawriter_protocol_status** (const **dds::core::InstanceHandle** &subscription_handle)

  *<<**extension**>> (p. 354) Get the datawriter protocol status for this writer, per matched subscription identified by the subscription_handle.*

- **rti::core::status::DataWriterProtocolStatus matched_subscription_datawriter_protocol_status** (const **rti::core::Locator** &subscription_locator)

  <<***extension***>> *(p. 354) Get the datawriter protocol status for this writer, per matched subscription identified by the locator*
- **rti::core::status::ServiceRequestAcceptedStatus service_request_accepted_status** ()

  <<***extension***>> *(p. 354) Get the service request accepted status for this writer*
- void **flush** ()

  <<***extension***>> *(p. 354) Flushes the batch in progress in the context of the calling thread.*
- void **write** (const T &instance_data, **rti::pub::WriteParams** &params)

  <<***extension***>> *(p. 354) Write with advanced parameters*
- **rti::core::Result**< void > **write_noexcept** (const T &sample) noexcept

  <<***extension***>> *(p. 354) Alternative to **write()** (p. 1148) that doesn't throw exceptions.*
- **rti::core::Result**< void > **write_noexcept** (const T &sample, const **dds::core::Time** &timestamp) noexcept

  <<***extension***>> *(p. 354) Alternative to **write()** (p. 1148) that doesn't throw exceptions.*
- **rti::core::Result**< void > **write_noexcept** (const T &sample, const **dds::core::InstanceHandle** &instance) noexcept

  <<***extension***>> *(p. 354) Alternative to **write()** (p. 1148) that doesn't throw exceptions.*
- **rti::core::Result**< void > **write_noexcept** (const T &sample, const **dds::core::InstanceHandle** &instance, const **dds::core::Time** &timestamp) noexcept

  <<***extension***>> *(p. 354) Alternative to **write()** (p. 1148) that doesn't throw exceptions.*
- **rti::core::Result**< void > **write_noexcept** (const T &instance_data, **rti::pub::WriteParams** &params) noexcept

  <<***extension***>> *(p. 354) Alternative to **write()** (p. 1148) that doesn't throw exceptions.*
- T * **create_data** ()

  <<***extension***>> *(p. 354) Create data from this writer and initialize it*
- bool **delete_data** (T *sample)

  <<***extension***>> *(p. 354) Delete data created with **create_data()** (p. 1182).*
- T * **get_loan** ()

  <<***extension***>> *(p. 354) Get a loaned sample from this writer*
- void **discard_loan** (T &sample)

  <<***extension***>> *(p. 354) Discard a loaned sample from this writer*
- const **dds::core::InstanceHandle register_instance** (const T &key, **rti::pub::WriteParams** &params)

  <<***extension***>> *(p. 354) Registers and instance with parameters*

Related Functions

(Note that these are not member functions.)

- void **ignore** (**dds::domain::DomainParticipant** &participant, const **dds::core::InstanceHandle** &handle)

  Instructs RTI Connext to locally ignore a publication.
- template<typename FwdIterator >
  void **ignore** (**dds::domain::DomainParticipant** &participant, FwdIterator begin, FwdIterator end)

  Instructs RTI Connext to locally ignore several publications.
- template<typename T >
  dds::core::InstanceHandleSeq matched_subscriptions (const **dds::pub::DataWriter**< T > &writer)

  *Retrieve the list of subscriptions currently associated with a **dds::pub::DataWriter** (p. 1138).*

- template<typename T , typename FwdIterator >
  FwdIterator **matched_subscriptions** (const **dds::pub::DataWriter**< T > &writer, FwdIterator begin, FwdIterator end)

  *Retrieve the list of subscriptions currently associated with a **dds::pub::DataWriter*** (p. *1138*).

- template<typename T >
  const **dds::topic::SubscriptionBuiltinTopicData** **matched_subscription_data** (const **dds::pub::DataWriter**< T > &writer, const **dds::core::InstanceHandle** &subscription_handle)

  *Retrieves information on a subscription that is currently associated with a **dds::pub::DataWriter*** (p. *1138*).

- template<typename T >
  dds::topic::ParticipantBuiltinTopicData **matched_subscription_participant_data** (const **dds::pub::DataWriter**< T > &writer, const **dds::core::InstanceHandle** &handle)

  <<***extension***>> (p. *354*) *This operation retrieves the information on the discovered **dds::domain::DomainParticipant*** (p. *1326*) *associated with the subscription that is currently matching with the **dds::pub::DataWriter*** (p. *1138*).

- template<typename T >
  rti::core::LocatorSeq **matched_subscriptions_locators** (const **dds::pub::DataWriter**< T > &writer)

  <<***extension***>> (p. *354*) *Retrieve the list of locators for subscriptions currently associated with this **DataWriter*** (p. *1138*).

- template<typename T >
  bool **is_matched_subscription_active** (const **dds::pub::DataWriter**< T > &writer, const **dds::core::InstanceHandle** &handle)

  <<***extension***>> (p. *354*) *Check if a matched subscription is active.*

- template<typename T >
  std::vector< **dds::topic::SubscriptionBuiltinTopicData** > **matched_subscription_data** (const **dds::pub::DataWriter**< T > &writer)

  <<***extension***>> (p. *354*) *Obtain the SubscriptionBuiltinTopicData for all of the subscriptions matched with a **DataWriter*** (p. *1138*).

- template<typename Writer , typename FwdIterator >
  uint32_t **find** (const **dds::pub::Publisher** & **publisher**, RTI_STRING_IN topic_name, FwdIterator begin, uint32_t max_size)

  *Retrieves the **dds::pub::DataWriter*** (p. *1138*) *for a specific topic name.*

- template<typename Writer , typename BinIterator >
  uint32_t **find** (const **dds::pub::Publisher** & **publisher**, RTI_STRING_IN topic_name, BinIterator begin)

  *Retrieves the **dds::pub::DataWriter*** (p. *1138*) *for a specific topic name.*

- template<typename AnyDataWriterBackInsertIterator >
  uint32_t **find_datawriters** (**dds::pub::Publisher** **publisher**, AnyDataWriterBackInsertIterator begin)

  *Retrieve all the **dds::pub::DataWriter*** (p. *1138*) *created from this **dds::pub::Publisher*** (p. *1993*).

- template<typename AnyDataWriterForwardIterator >
  uint32_t **find_datawriters** (**dds::pub::Publisher** **publisher**, AnyDataWriterForwardIterator begin, uint32_t max_size)

  *Retrieve all the **dds::pub::DataWriter*** (p. *1138*) *created from this **dds::pub::Publisher*** (p. *1993*).

- template<typename Writer >
  Writer **find_datawriter_by_topic_name** (**dds::pub::Publisher** **publisher**, const std::string &topic_name)

  <<***extension***>> (p. *354*) *Retrieves a **dds::pub::DataWriter*** (p. *1138*) *with the given name within the **dds::pub::Publisher*** (p. *1993*)

- template<typename Writer >
  Writer **find_datawriter_by_name** (**dds::pub::Publisher** **publisher**, const std::string &datawriter_name)

  <<***extension***>> (p. *354*) *Retrieves a **dds::pub::DataWriter*** (p. *1138*) *with the given name within the **dds::pub::Publisher*** (p. *1993*)

- template<typename Writer >
  Writer **find_datawriter_by_name** (**dds::domain::DomainParticipant** participant, const std::string &datawriter_name)

*<<**extension**>> (p. 354) Retrieves a **dds::pub::DataWriter** (p. 1138) within the **dds::domain::DomainParticipant** (p. 1326) with the given name*

8.67.1 Detailed Description

template<typename T>
class dds::pub::DataWriter< T >

*<<**reference-type**>> (p. 351)* Allows an application to publish data for a **dds::topic::Topic** (p. 2474)

Template Parameters

T	The topic-type that the **DataWriter** (p. 1138) publishes

QoS:

dds::pub::qos::DataWriterQos (p. 1226)

Status:

dds::core::status::StatusMask::liveliness_lost() (p. 2383), **dds::core::status::LivelinessLostStatus** (p. 1655);
dds::core::status::StatusMask::offered_deadline_missed() (p. 2379), **dds::core::status::OfferedDeadlineMissedStatus** (p. 1871);
dds::core::status::StatusMask::offered_incompatible_qos() (p. 2380), **dds::core::status::OfferedIncompatibleQosStatus** (p. 1873);
dds::core::status::StatusMask::publication_matched() (p. 2384), **dds::core::status::PublicationMatchedStatus** (p. 1990);
dds::core::status::StatusMask::reliable_reader_activity_changed() (p. 2385), **rti::core::status::ReliableReaderActivityChangedStatus** (p. 2157);
dds::core::status::StatusMask::reliable_writer_cache_changed() (p. 2385), **rti::core::status::ReliableWriterCacheChangedStatus** (p. 2159).

Listener:

dds::pub::DataWriterListener (p. 1202)

A **dds::pub::DataWriter** (p. 1138) is attached to exactly one **dds::pub::Publisher** (p. 1993), that acts as a factory for it.

A **dds::pub::DataWriter** (p. 1138) is bound to exactly one **dds::topic::Topic** (p. 2474) and therefore to exactly one data type. The **dds::topic::Topic** (p. 2474) must exist prior to the **dds::pub::DataWriter** (p. 1138)'s creation.

The following operations may be called even if the **dds::pub::DataWriter** (p. 1138) is not enabled. Other operations will fail with **dds::core::NotEnabledError** (p. 1869) if called on a disabled **dds::pub::DataWriter** (p. 1138):

- **dds::pub::DataWriter::qos(const dds::pub::qos::DataWriterQos&)** (p. 1166), **dds::pub::DataWriter::qos() const** (p. 1166), **dds::pub::DataWriter::set_listener** (p. 1169), **dds::pub::DataWriter::get_listener** (p. 1170), **dds::core::Entity::enable** (p. 1524), **dds::core::cond::StatusCondition::StatusCondition(const dds::core::Entity&)** (p. 2373) and **dds::core::Entity::status_changes()** (p. 1525) **dds::pub::DataWriter::liveliness_lost_status()** (p. 1171), **dds::pub::DataWriter::offered_deadline_missed_status()** (p. 1171), **dds::pub::DataWriter::offered_incompatible_qos_status()** (p. 1171), **dds::pub::DataWriter::publication_matched_status()** (p. 1172), **dds::pub::DataWriter::reliable_writer_cache_changed_status()** (p. 1175), **dds::pub::DataWriter::reliable_reader_activity_changed_status()** (p. 1175) **dds::pub::DataWriter::service_request_accepted_status()** (p. 1178)

Several **dds::pub::DataWriter** (p. 1138) may operate in different threads. If they share the same **dds::pub::Publisher** (p. 1993), the middleware guarantees that its operations are thread-safe.

See also

Operations Allowed in Listener Callbacks (p. 1639)

8.67.2 Notes about DataWriter destruction

The deletion of the **dds::pub::DataWriter** (p. 1138) will automatically unregister all instances.

The **DataWriter** (p. 1138) destructor will not throw any exceptions even if the destruction fails. Calling **close()** (p. 1525) will throw in case of failure

See also

DataWriter Use Cases (p. 311)

Entity Use Cases (p. 325)

Examples

Foo_publisher.cxx.

8.67.3 Constructor & Destructor Documentation

8.67.3.1 DataWriter() [1/6]

```
template<typename T >
dds::pub::DataWriter< T >::DataWriter (
            const  dds::pub::Publisher & pub,
            const  dds::topic::Topic< T > & the_topic )  [inline]
```

Creates a **DataWriter** (p. 1138).

It uses the default DataWriterQos, and sets no listener.

Parameters

pub	The publisher that this **DataWriter** (p. 1138) belongs to
the_topic	The Topic associated with this **DataWriter** (p. 1138)

See also

DataWriter(const dds::pub::Publisher& pub,const dds::topic::Topic<T>& topic,const dds::pub::qos::DataWriterQos& qos,dds::pub::DataWriterListener<T>* listener,const dds::core::status::StatusMask& mask) (p. 1145)

8.67.3.2 DataWriter() [2/6]

```
template<typename T >
dds::pub::DataWriter< T >::DataWriter (
             const  dds::pub::Publisher & pub,
             const  dds::topic::Topic< T > & the_topic,
             const  dds::pub::qos::DataWriterQos & the_qos,
              dds::pub::DataWriterListener< T > * the_listener = NULL,
             const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Creates a **DataWriter** (p. 1138) with QoS and listener.

[DEPRECATED] When using a listener, prefer the constructor that receives a shared_ptr<**Listener** (p. 1637)> instead of a regular Listener* pointer.

When a **DataWriter** (p. 1138) is created, only those transports already registered are available to the **DataWriter** (p. 1138). See **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered.

Precondition

If publisher is enabled, topic must have been enabled. Otherwise, this operation will fail and no **DataWriter** (p. 1138) will be created.

The given **dds::topic::Topic** (p. 2474) must have been created from the same participant as this publisher. If it was created from a different participant, this method will fail.

Parameters

pub	The publisher that this **DataWriter** (p. 1138) belongs to
the_topic	The Topic associated with this **DataWriter** (p. 1138)
the_qos	QoS to be used for creating the new Datawriter.
the_listener	The **DataWriter** (p. 1138) listener. The caller owns the listener and is responsible for deleting it only after resetting it or after deleting the **DataWriter** (p. 1138).
mask	Changes of communication status to be invoked on the listener

See also

Specifying QoS on entities (p. 513) for information on setting QoS before entity creation

dds::pub::qos::DataWriterQos (p. 1226) for rules on consistency among QoS

dds::core::QosProvider::datawriter_qos() (p. 2035)

dds::pub::Publisher::default_datawriter_qos() (p. 1998)

dds::pub::qos::DataWriterQos::operator=(const dds::topic::qos::TopicQos&) (p. 1230) which allows assigning the contents of a TopicQos into a DataWriterQos

listener() (p. 1169)

8.67.3.3 DataWriter() [3/6]

```
template<typename T >
dds::pub::DataWriter< T >::DataWriter (
             const  dds::pub::Publisher & pub,
             const  dds::topic::Topic< T > & the_topic,
             const  dds::pub::qos::DataWriterQos & the_qos,
             std::shared_ptr<  Listener > the_listener,
             const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Creates a **DataWriter** (p. 1138) with QoS and listener.

When a **DataWriter** (p. 1138) is created, only those transports already registered are available to the **DataWriter** (p. 1138). See **Built-in Transport Plugins** (p. 83) for details on when a builtin transport is registered.

Precondition

If publisher is enabled, topic must have been enabled. Otherwise, this operation will fail and no **DataWriter** (p. 1138) will be created.

The given **dds::topic::Topic** (p. 2474) must have been created from the same participant as this publisher. If it was created from a different participant, this method will fail.

Parameters

pub	The publisher that this **DataWriter** (p. 1138) belongs to
the_topic	The Topic associated with this **DataWriter** (p. 1138)
the_qos	QoS to be used for creating the new Datawriter.
the_listener	A shared_ptr to the listener. See **set_listener()** (p. 1169) for more information.
mask	Indicates which status updates the listener will receive

See also

Specifying QoS on entities (p. 513) for information on setting QoS before entity creation

dds::pub::qos::DataWriterQos (p. 1226) for rules on consistency among QoS

dds::core::QosProvider::datawriter_qos() (p. 2035)

dds::pub::Publisher::default_datawriter_qos() (p. 1998)

dds::pub::qos::DataWriterQos::operator=(const dds::topic::qos::TopicQos&) (p. 1230) which allows assigning the contents of a TopicQos into a DataWriterQos

listener() (p. 1169)

8.67.3.4 DataWriter() [4/6]

```
template<typename T >
dds::pub::DataWriter< T >::DataWriter (
            const  dds::topic::Topic< T > & topic )  [inline], [explicit]
```

Creates a **DataWriter** (p. 1138).

It uses the default DataWriterQos, and sets no listener.

This constructor uses the implicit publisher to create this **DataWriter** (p. 1138) (**rti::pub::implicit_↩ publisher** (p. 738)(topic.participant())).

Parameters

topic	The Topic associated with this **DataWriter** (p. 1138)

See also

DataWriter(const dds::pub::Publisher& pub,const dds::topic::Topic<T>& topic,const dds::pub::qos↩ ::DataWriterQos& qos,dds::pub::DataWriterListener<T>* listener,const dds::core::status::StatusMask& mask) (p. 1145)

8.67.3.5 DataWriter() [5/6]

```
template<typename T >
dds::pub::DataWriter< T >::DataWriter (
            const  dds::topic::Topic< T > & topic,
            const  dds::pub::qos::DataWriterQos & qos )  [inline]
```

Creates a **DataWriter** (p. 1138) with QoS.

This constructor uses the implicit publisher to create this **DataWriter** (p. 1138)

Parameters

topic	The Topic associated with this **DataWriter** (p. 1138)
qos	QoS to be used for creating the new Datawriter.

See also

DataWriter(const dds::pub::Publisher& pub,const dds::topic::Topic<T>& topic,const dds::pub::qos::DataWriterQos& qos,dds::pub::DataWriterListener<T>* listener,const dds::core::status::StatusMask& mask) (p. 1145)

8.67.3.6 DataWriter() [6/6]

```
template<typename T >
dds::pub::DataWriter< T >::DataWriter (
            const  dds::topic::Topic< T > & topic,
            const  dds::pub::qos::DataWriterQos & qos,
            std::shared_ptr<  Listener > listener,
            const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Creates a **DataWriter** (p. 1138) with QoS and listener.

This constructor uses the implicit publisher to create this **DataWriter** (p. 1138)

Parameters

topic	The Topic associated with this **DataWriter** (p. 1138)
qos	QoS to be used for creating the new Datawriter.
listener	A shared_ptr to the listener. See **set_listener()** (p. 1169) for more information.
mask	Indicates which status updates the listener will receive

See also

DataWriter(const dds::pub::Publisher& pub,const dds::topic::Topic<T>& topic,const dds::pub::qos::DataWriterQos& qos,dds::pub::DataWriterListener<T>* listener,const dds::core::status::StatusMask& mask) (p. 1145)

8.67.4 Member Function Documentation

8.67.4.1 write() [1/11]

```
template<typename T >
void  dds::pub::DataWriter< T >::write (
             const T & instance_data )  [inline]
```

Modifies the value of a data instance.

This operations does the same as **write(const T&, const dds::core::InstanceHandle&)** (p. 1149) except that it deduces the identity of the instance from `instance_data` (by means of the key).

Parameters

instance_data	The data sample to write.

See also

write(const T&, const dds::core::InstanceHandle&) (p. 1149)

Examples

Foo_publisher.cxx.

8.67.4.2 write() [2/11]

```
template<typename T >
void  dds::pub::DataWriter< T >::write (
             const T & instance_data,
             const  dds::core::Time & timestamp )  [inline]
```

Modifies the value of a data instance and specifies the timestamp.

See also

void **write(const T&, const dds::core::InstanceHandle&, const dds::core::Time&)** (p. 1153)

Notes about DataWriter destruction (p. 1144)

8.67.4.3 write() [3/11]

```
template<typename T >
void  dds::pub::DataWriter< T >::write (
            const T & instance_data,
            const  dds::core::InstanceHandle & handle )  [inline]
```

Modifies the value of a data instance.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is made available to **dds::sub::DataReader** (p. 976) objects by means of the `source_timestamp` attribute inside the **dds::sub::SampleInfo** (p. 2285). (Refer to **dds::sub::SampleInfo** (p. 2285) and **DESTINATION_ORDER** (p. 524) QoS policy for details).

As a side effect, this operation asserts liveliness on the **dds::pub::DataWriter** (p. 1138) itself, the **dds::pub::Publisher** (p. 1993) and the **dds::domain::DomainParticipant** (p. 1326).

Note that the special value **dds::core::InstanceHandle::nil()** (p. 1614) can be used for the parameter `handle`. This indicates the identity of the instance should be automatically deduced from the `instance_data` (by means of the `key`).

If `handle` is any value other than **dds::core::InstanceHandle::nil()** (p. 1614), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **dds::core::InvalidArgumentError** (p. 1619).

RTI Connext will not detect the error when the handle is any value other than **dds::core::InstanceHandle::nil()** (p. 1614), corresponds to an instance that has been registered, but does not correspond to the instance deduced from the `instance_data` (by means of the `key`). RTI Connext will treat as if the **write()** (p. 1148) operation is for the instance as indicated by the `handle`.

This operation may block if the **RELIABILITY** (p. 544) `kind` is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) and the modification would cause data to be lost or else cause one of the limits specified in the **RESOURCE_LIMITS** (p. 546) to be exceeded.

This operation will not block when using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157). If you are using BEST_EFFORT Reliability in combination with **rti::core::policy::PublishModeKind_def::ASYNCHRONOUS** (p. 2019), then instead of being blocked, samples that are queued to be sent by the asynchronous publishing thread will be overwritten when the number of DDS samples that are currently queued has reached the `depth` QoS value in the **dds::core::policy::History** (p. 1602).

If **dds::core::policy::Reliability::max_blocking_time** (p. 2153) elapses before the **dds::pub::DataWriter** (p. 1138) can store the modification without exceeding the limits, the operation will fail and return **dds::core::TimeoutError** (p. 2473) for KEEP_ALL configurations.

Here is how the write operation behaves when **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607) and **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) are used:

- The send window size is determined by the **rti::core::policy::RtpsReliableWriterProtocol::max_send_window_size** (p. 2250) and **rti::core::policy::RtpsReliableWriterProtocol::min_send_window_size** (p. 2249) fields in the **rti::core::policy::DataWriterProtocol** (p. 1210). If a send window is specified (max_send_window_size is not UNLIMITED) and the window is full, the write operation will block until one of the samples in the send window is protocol-acknowledged (ACKed) (1) or until the **dds::core::policy::Reliability::max_blocking_time** (p. 2153) expires.

- Then, the **dds::pub::DataWriter** (p. 1138) will try to add the new sample to the writer history.
- If the instance associated with the sample is present in the writer history and there are `depth` (in the **HISTORY** (p. 533)) samples in the instance, the **DataWriter** (p. 1138) will replace the oldest sample of that instance independently of that sample's acknowledged status, and the write operation will return . Otherwise, no sample will be replaced and the write operation will continue.
- If the instance associated with the sample is not present in the writer history and **dds::core::policy::Resource↩ Limits::max_instances** (p. 2206) is exceeded, the **DataWriter** (p. 1138) will try to replace an existing instance (and its samples) according to the value of **rti::core::policy::DataWriterResourceLimits::instance↩ _replacement** (p. 1241) (see DataWriterResourceLimitsInstanceReplacementKind).
  - If no instance can be replaced, the write operation returns **dds::core::OutOfResourcesError** (p. 1898).
- If **dds::core::policy::ResourceLimits::max_samples** (p. 2205) is exceeded, the **DataWriter** (p. 1138) will try to drop a sample from a different instance as follows:
  - The **DataWriter** (p. 1138) will try first to remove a fully ACKed (2) sample from a different instance 'I' as long as that sample is not the last remaining sample for the instance 'I'. To find this sample, the **DataWriter** (p. 1138) starts iterating from the oldest sample in the writer history to the newest sample.
  - If no such sample is found, the **DataWriter** (p. 1138) will replace the oldest sample in the writer history.
- The sample is added to the writer history, and the write operation returns .

Here is how the write operation behaves when **dds::core::policy::HistoryKind_def::KEEP_ALL** (p. 1607) and **dds↩ ::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) are used:

- The send window size is determined by the **rti::core::policy::RtpsReliableWriterProtocol::max_send_↩ window_size** (p. 2250) and **rti::core::policy::RtpsReliableWriterProtocol::min_send_window_size** (p. 2249) fields in the **DATA_WRITER_PROTOCOL** (p. 521). If a send window is specified (max_send_window_size is not UNLIMITED) and the window is full, the write operation will block until one of the samples in the send window is protocol-acknowledged (ACKed) (1) or until the **dds::core::policy::Reliability::max_blocking_time** (p. 2153) expires.
  - If the max_blocking_time expires, the write operation returns **dds::core::TimeoutError** (p. 2473).
- When a sample is protocol-ACKed (1) before max_blocking_time expires, the **DataWriter** (p. 1138) will try to add the sample to the writer history as follows:
  - If the instance associated with the sample is not present in the writer history and max_instances is exceeded, the **DataWriter** (p. 1138) will try to replace an existing instance (and its samples) according to the value of **rti::core::policy::DataWriterResourceLimits::instance_replacement** (p. 1241) (see Data↩ WriterResourceLimitsInstanceReplacementKind).
    - If no instance can be replaced, the write operation returns **dds::core::OutOfResourcesError** (p. 1898).
  - If **dds::core::policy::ResourceLimits::max_samples** (p. 2205) is exceeded, the **DataWriter** (p. 1138) will go through the samples in the order in which they were added, and it will replace the first sample that is fully ACKed (2).
    - If no fully ACKed sample is found, the **DataWriter** (p. 1138) will block (3) until a sample is fully ACKed and can be replaced or **dds::core::policy::Reliability::max_blocking_time** (p. 2153) expires. If the max_blocking_time expires, the write operation will return **dds::core::TimeoutError** (p. 2473).
  - If **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207) is exceeded, the **Data↩ Writer** (p. 1138) will go through the samples of the instance in the order in which they were added, and it will replace the first sample that is fully ACKed.

    - If no fully ACKed sample is found, the **DataWriter** (p. 1138) will block (3) until a sample is fully ACKed and can be replaced or the max_blocking_time expires. If the max_blocking_time expires, the write operation will return **dds::core::TimeoutError** (p. 2473).
  - The sample is added to the writer history, and the write operation returns .

If there are no instance resources left, this operation may fail with **dds::core::OutOfResourcesError** (p. 1898). Calling **dds::pub::DataWriter::unregister_instance** (p. 1160) may help freeing up some resources.

This operation will fail with **dds::core::PreconditionNotMetError** (p. 1941) if the timestamp is less than the timestamp used in the last writer operation (*register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application-provided timestamp).

If an instance obtained from **dds::pub::DataWriter::get_loan** (p. 1183) is modified with this operation, then all instances modified thereafter should be from **dds::pub::DataWriter::get_loan** (p. 1183).

See **rti::core::policy::AcknowledgmentKind_def** (p. 797) for more information on the following notes:

(1) A sample in the writer history is considered "protocol ACKed" when the sample has been individually ACKed at the RTPS protocol level by each one of the DataReaders that matched the **DataWriter** (p. 1138) at the moment the sample was added to the writer queue.

- Late joiners do not change the protocol ACK state of a sample. If a sample is marked as protocol ACKed because it has been acknowledged by all the matching DataReaders and a DataReader joins later on, the historical sample is still considered protocol ACKed even if it has not been received by the late joiner.
- If a sample 'S1' is protocol ACKed and a TopicQuery is received, triggering the publication of 'S1', the sample is still considered protocol ACKed. If a sample 'S1' is not ACKed and a TopicQuery is received triggering the publication of 'S1', the **DataWriter** (p. 1138) will require that both the matching DataReaders on the live RTPS channel and the DataReader on the TopicQuery channel individually protocol ACK the sample in order to consider the sample protocol ACKed.

(2) A sample in the writer history is considered "fully ACKed" when all of the following conditions are met:

- The sample is protocol-ACKed.
- The sample has been "application-level ACKed" by all the DataReaders matching the **DataWriter** (p. 1138) that have their **dds::core::policy::Reliability::acknowledgment_kind** (p. 2154) set to **rti::core::policy::AcknowledgmentKind_def::APPLICATION_EXPLICIT** (p. 798) or **rti::core::policy::AcknowledgmentKind_def::APPLICATION_AUTO** (p. 798). Once the sample is application-level ACKed, it cannot change its status to not ACKed after new DataReaders are matched. (Application-level ACK occurs when the application acknowledges receipt of a sample.)
- If required subscriptions are enabled (see **rti::core::policy::Availability** (p. 868)), the sample must also be ACKed by all the required subscriptions configured on the **DataWriter** (p. 1138).

(3) It is possible within a single call to the write operation for a **DataWriter** (p. 1138) to block both when the send window is full and then again when **dds::core::policy::ResourceLimits::max_samples** (p. 2205) or **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207) is exceeded. This can happen because blocking on the send window only considers protocol-ACKed samples, while blocking based on resource limits considers fully-ACKed samples. In any case, the total max blocking time of a single call to the write operation will not exceed **dds::core::policy::Reliability::max_blocking_time** (p. 2153).

Parameters

instance_data	<<***in***>> *(p. 355)* The data to write.

Parameters

handle	<<***in***>> *(p. 355)* Either the handle returned by a previous call to **dds::pub::DataWriter::register_instance** (p. 1158), or else the special value **dds::core::InstanceHandle::nil()** (p. 1614). If `T` has a key, `handle` must represent a registered instance of type `T`. Otherwise, this method may fail with **dds::core::InvalidArgumentError** (p. 1619).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::TimeoutError** (p. 2473), **dds::core::PreconditionNotMetError** (p. 1941), **dds::core::OutOfResourcesError** (p. 1898), or **dds::core::NotEnabledError** (p. 1869).

MT Safety:

It is UNSAFE to modify `instance_data` before the operation has finished. The operation is otherwise SAFE.

See also

dds::sub::DataReader (p. 976)

dds::pub::DataWriter::write(const T&,const dds::core::Time&) (p. 1149)

DESTINATION_ORDER (p. 524)

8.67.4.4 write() [4/11]

```
template<typename T >
void  dds::pub::DataWriter< T >::write (
             const T & instance_data,
             const  dds::core::InstanceHandle & handle,
             const  dds::core::Time & source_timestamp )  [inline]
```

Modifies the value of a data instance and specifies the timestamp.

Explicitly provides the timestamp that will be available to the **dds::sub::DataReader** (p. 976) objects by means of the `source_timestamp` attribute inside the **dds::sub::SampleInfo** (p. 2285). (Refer to **dds::sub::SampleInfo** (p. 2285) and **DESTINATION_ORDER** (p. 524) QoS policy for details)

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **dds::pub::DataWriter::write()** (p. 1148) operation.

This operation may block and time out (**dds::core::TimeoutError** (p. 2473)) under the same circumtances described for **dds::pub::DataWriter::write()** (p. 1148).

If there are no instance resources left, this operation may fail with **dds::core::OutOfResourcesError** (p. 1898). Calling **dds::pub::DataWriter::unregister_instance** (p. 1160) may help free up some resources.

This operation may fail with **dds::core::InvalidArgumentError** (p. 1619) under the same circumstances described for the write operation.

Parameters

instance_data	<<***in***>> *(p. 355)* The data to write.
handle	<<***in***>> *(p. 355)* Either the handle returned by a previous call to **dds::pub::DataWriter::register_instance** (p. 1158), or else the special value **dds::core::InstanceHandle::nil()** (p. 1614). If `T` has a key, `handle` must represent a registered instance of type `T`. Otherwise, this method may fail with **dds::core::InvalidArgumentError** (p. 1619).
source_timestamp	<<***in***>> *(p. 355)* When using **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264) the timestamp value must be greater than or equal to the timestamp value used in the last writer operation (*register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application-provided timestamp) However, if it is less than the timestamp of the previous operation but the difference is less than the **dds::core::policy::DestinationOrder::source_timestamp_tolerance** (p. 1262), the timestamp of the previous operation will be used as the source timestamp of this sample. Otherwise, if the difference is greater than **dds::core::policy::DestinationOrder::source_timestamp_tolerance** (p. 1262), the function will return **dds::core::InvalidArgumentError** (p. 1619).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::TimeoutError** (p. 2473), **dds::core::OutOfResourcesError** (p. 1898), or **dds::core::NotEnabledError** (p. 1869).

See also

dds::pub::DataWriter::write() (p. 1148)

dds::sub::DataReader (p. 976)

DESTINATION_ORDER (p. 524)

Notes about DataWriter destruction (p. 1144)

8.67.4.5 write() [5/11]

```
template<typename T >
void  dds::pub::DataWriter< T >::write (
            const  dds::topic::TopicInstance< T > & topic_instance )  [inline]
```

Write a **dds::topic::TopicInstance** (p. 2505).

A TopicInstance encapsulates the sample and its associated instance handle.

Parameters

topic_instance	The instance to write.

See also

write(const T&, const dds::core::InstanceHandle&) (p. 1149)

8.67.4.6 write() [6/11]

```
template<typename T >
void  dds::pub::DataWriter< T >::write (
            const  dds::topic::TopicInstance< T > & topic_instance,
            const  dds::core::Time & timestamp )  [inline]
```

Write a topic instance with time stamp.

Parameters

topic_instance	the TopicInstance to write.
timestamp	the timestamp for this sample.

See also

void **write(const T&, const dds::core::InstanceHandle&, const dds::core::Time&)** (p. 1153)

Notes about DataWriter destruction (p. 1144)

8.67.4.7 write() [7/11]

```
template<typename T >
template<typename FWIterator >
void  dds::pub::DataWriter< T >::write (
            const FWIterator & begin,
            const FWIterator & end )  [inline]
```

Write a series of samples or TopicInstances

Template Parameters

FWIterator	A forward iterator. Depending on its value type this function can write data samples or TopicInstance objects.

Parameters

begin	The beginning of the range
end	The end of the range

See also

write(const T&) (p. 1148)

write(const dds::topic::TopicInstance<T>&) (p. 1154)

8.67.4.8 write() [8/11]

```
template<typename T >
template<typename FWIterator >
void  dds::pub::DataWriter< T >::write (
            const FWIterator & begin,
            const FWIterator & end,
            const  dds::core::Time & timestamp )  [inline]
```

Write a series of samples or TopicInstances with a given timestamp

Template Parameters

FWIterator	A forward iterator. Depending on its value type this function can write data samples or TopicInstance objects.

Parameters

begin	The beginning of the range
end	The end of the range
timestamp	The timestamp to use for all samples in the range

See also

write(const T&, const dds::core::Time&) (p. 1149)

write(const dds::topic::TopicInstance<T>&, const dds::core::Time&) (p. 1155)

Notes about DataWriter destruction (p. 1144)

8.67.4.9 write() [9/11]

```
template<typename T >
template<typename SamplesFWIterator , typename HandlesFWIterator >
void  dds::pub::DataWriter< T >::write (
            const SamplesFWIterator & data_begin,
            const SamplesFWIterator & data_end,
            const HandlesFWIterator & handle_begin,
            const HandlesFWIterator & handle_end )  [inline]
```

Write a series of samples and their parallel instance handles.

Template Parameters

SamplesFWIterator	Sample forward iterator
HandlesFWIterator	InstanceHandle forward iterator

Parameters

data_begin	The beginning of the data sample range
data_end	The end of the data sample range
handle_begin	The beginning of the InstanceHandle range
handle_end	The end of the InstanceHandle range

See also

write(const T&, const dds::core::InstanceHandle&) (p. 1149)

8.67.4.10 write() [10/11]

```
template<typename T >
template<typename SamplesFWIterator , typename HandlesFWIterator >
void  dds::pub::DataWriter< T >::write (
             const SamplesFWIterator & data_begin,
             const SamplesFWIterator & data_end,
             const HandlesFWIterator & handle_begin,
             const HandlesFWIterator & handle_end,
             const  dds::core::Time & timestamp )  [inline]
```

Write a series of samples and their parallel instance handles and a timestamp.

See also

Notes about DataWriter destruction (p. 1144)

8.67.4.11 operator<<() [1/4]

```
template<typename T >
DataWriter &  dds::pub::DataWriter< T >::operator<< (
             const T & data )  [inline]
```

Writes a sample.

See also

write(const T&) (p. 1148)

8.67.4.12 operator<<() [2/4]

```
template<typename T >
DataWriter &  dds::pub::DataWriter< T >::operator<< (
             const std::pair< T,  dds::core::Time > & data )  [inline]
```

Writes a sample with a timestamp.

See also

write(const T&, const dds::core::Time&) (p. 1149)

Notes about DataWriter destruction (p. 1144)

8.67.4.13 operator<<() [3/4]

```
template<typename T >
DataWriter &  dds::pub::DataWriter< T >::operator<< (
             const std::pair< T,  dds::core::InstanceHandle > & data )  [inline]
```

Writes a sample with an instance handle.

See also

write(const T&, const dds::core::InstanceHandle&) (p. 1149)

8.67.4.14 register_instance() [1/3]

```
template<typename T >
const  dds::core::InstanceHandle  dds::pub::DataWriter< T >::register_instance (
             const T & instance_data )  [inline]
```

Informs RTI Connext that the application will be modifying a particular instance.

This operation is only useful for keyed data types. Using it for non-keyed types causes no effect and returns **dds::core::InstanceHandle::nil()** (p. 1614). The operation takes as a parameter an instance (of which only the key value is examined) and returns a `handle` that can be used in successive **write()** (p. 1148) or **dispose_instance()** (p. 1162) operations.

The operation gives RTI Connext an opportunity to pre-configure itself to improve performance.

The use of this operation by an application is optional even for keyed types. If an instance has not been pre-registered, the application can call the **write()** (p. 1148) or dipose_instance() overloads that don't receive a **dds::core::InstanceHandle** (p. 1612) and RTI Connext will auto-register the instance.

For best performance, the operation should be invoked prior to calling any operation that modifies the instance, such as **dds::pub::DataWriter::write()** (p. 1148), **dds::pub::DataWriter::write(const T&,const dds::core::Time&)** (p. 1149), **dds::pub::DataWriter::dispose_instance()** (p. 1162) and **dds::pub::DataWriter::dispose_instance(const dds::core::InstanceHandle&,const dds::core::Time&)** (p. 1163) and the handle used in conjunction with the data for those calls.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is used.

This operation may fail and return **dds::core::InstanceHandle::nil()** (p. 1614) if **dds::core::policy::ResourceLimits::max_instances** (p. 2206) limit has been exceeded.

The operation is **idempotent**. If it is called for an already registered instance, it just returns the already allocated handle. This may be used to lookup and retrieve the handle allocated to a given instance.

This operation can only be called after **dds::pub::DataWriter** (p. 1138) has been enabled. Otherwise, **dds::core::InstanceHandle::nil()** (p. 1614) will be returned.

Parameters

instance_data	<<***in***>> *(p. 355)* The instance that should be registered. Of this instance, only the fields that represent the key are examined by the function.

Returns

For keyed data type, a handle that can be used in the calls that take a **dds::core::InstanceHandle** (p. 1612), such as write, dispose, unregister_instance, or return **dds::core::InstanceHandle::nil()** (p. 1614) on failure. If the instance_data is of a data type that has no keys, this function always returns **dds::core::InstanceHandle::nil()** (p. 1614).

See also

dds::pub::DataWriter::unregister_instance (p. 1160), **dds::pub::DataWriter::key_value** (p. 1164), **Relationship between registration, liveliness and ownership** (p. 1902)

8.67.4.15 register_instance() [2/3]

```
template<typename T >
const  dds::core::InstanceHandle  dds::pub::DataWriter< T >::register_instance (
             const T & instance_data,
             const  dds::core::Time & source_timestamp )  [inline]
```

Informs RTI Connext that the application will be modifying a particular instance and specifies the timestamp.

The provided `source_timestamp` potentially affects the relative order in which readers observe events from multiple writers. Refer to **DESTINATION_ORDER** (p. 524) QoS policy for details.

This operation may fail and return **dds::core::InstanceHandle::nil()** (p. 1614) if **dds::core::policy::ResourceLimits::max_instances** (p. 2206) limit has been exceeded.

This operation can only be called after **dds::pub::DataWriter** (p. 1138) has been enabled. Otherwise, **dds::core::InstanceHandle::nil()** (p. 1614) will be returned.

Parameters

instance_data	<<***in***>> *(p. 355)* The instance that should be registered. Of this instance, only the fields that represent the key are examined by the function.
source_timestamp	<<***in***>> *(p. 355)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers.

Returns

For keyed data type, return a handle that can be used in the calls that take a **dds::core::InstanceHandle** (p. 1612), such as write, dispose, unregister_instance, or return **dds::core::InstanceHandle::nil()** (p. 1614) on failure. If the instance_data is of a data type that has no keys, this function always return **dds::core::InstanceHandle::nil()** (p. 1614).

See also

dds::pub::DataWriter::unregister_instance (p. 1160), **dds::pub::DataWriter::key_value** (p. 1164)

Notes about DataWriter destruction (p. 1144)

8.67.4.16 unregister_instance() [1/3]

```
template<typename T >
DataWriter &  dds::pub::DataWriter< T >::unregister_instance (
             const  dds::core::InstanceHandle & handle )  [inline]
```

Unregister an instance.

This operation is useful only for keyed data types. Using it for non-keyed types causes no effect and reports no error.

This operation should only be called on an instance that is currently registered. This includes instances that have been auto-registered by calling operations such as write or dispose as described in **dds::pub::DataWriter::register_instance** (p. 1158). Otherwise, this operation may fail with **dds::core::InvalidArgumentError** (p. 1619).

This only need be called just once per instance, regardless of how many times register_instance was called for that instance.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is used.

This operation informs RTI Connext that the **dds::pub::DataWriter** (p. 1138) is no longer going to provide any information about the instance. This operation also indicates that RTI Connext can locally remove all information regarding that instance. The application should not attempt to use the `handle` previously allocated to that instance after calling this function.

The parameter `handle` must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **dds::core::InvalidArgumentError** (p. 1619).

If, after a **dds::pub::DataWriter::unregister_instance** (p. 1160), the application wants to modify (**dds::pub::DataWriter::write()** (p. 1148) or **dds::pub::DataWriter::dispose_instance()** (p. 1162)) an instance, it has to register it again, or else use the functions that don't receive a **dds::core::InstanceHandle** (p. 1612).

This operation does not indicate that the instance is deleted (that is the purpose of **dds::pub::DataWriter::dispose_instance()** (p. 1162)). The operation **dds::pub::DataWriter::unregister_instance** (p. 1160) just indicates that the **dds::pub::DataWriter** (p. 1138) no longer has anything to say about the instance. **dds::sub::DataReader** (p. 976) entities that are reading the instance may receive a sample with **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) for the instance, unless there are other **dds::pub::DataWriter** (p. 1138) objects writing that same instance.

dds::core::policy::WriterDataLifecycle::autodispose_unregistered_instances (p. 2661) controls whether instances are automatically disposed when they are unregistered.

This operation can affect the ownership of the data instance (see **OWNERSHIP** (p. 538)). If the **dds::pub::DataWriter** (p. 1138) was the exclusive owner of the instance, then calling **unregister_instance()** (p. 1160) will relinquish that ownership.

If **dds::core::policy::Reliability::kind** (p. 2153) is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) and the unregistration would overflow the resource limits of this writer or of a reader, this operation may block for up to **dds::core::policy::Reliability::max_blocking_time** (p. 2153); if this writer is still unable to unregister after that period, this method will fail with **dds::core::TimeoutError** (p. 2473).

Parameters

handle	*<<**in**>> (p. 355)* represents the `instance` to be unregistered. It must represent an instance that has been registered. Otherwise, this method may fail with **dds::core::InvalidArgumentError** (p. 1619).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::TimeoutError** (p. 2473) or **dds::core::NotEnabledError** (p. 1869)

See also

dds::pub::DataWriter::register_instance (p. 1158)

dds::pub::DataWriter::unregister_instance(const dds::core::InstanceHandle&,const dds::core::Time&) (p. 1161)

dds::pub::DataWriter::key_value (p. 1164)

Relationship between registration, liveliness and ownership (p. 1902)

8.67.4.17 unregister_instance() [2/3]

```
template<typename T >
DataWriter & dds::pub::DataWriter< T >::unregister_instance (
```

```
const  dds::core::InstanceHandle & handle,
const  dds::core::Time & source_timestamp )  [inline]
```

Unregister an instance with timestamp.

The provided `source_timestamp` potentially affects the relative order in which readers observe events from multiple writers. Refer to **DESTINATION_ORDER** (p. 524) QoS policy for details.

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **dds::pub::DataWriter::unregister_instance** (p. 1160) operation.

This operation may block and may time out (**dds::core::TimeoutError** (p. 2473)) under the same circumtances described for the unregister_instance operation.

Parameters

handle	<<***in***>> *(p. 355)* represents the `instance` to be unregistered. It must represent an instance that has been registered. Otherwise, this method may fail with **dds::core::InvalidArgumentError** (p. 1619).
source_timestamp	<<***in***>> *(p. 355)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::TimeoutError** (p. 2473) or **dds::core::NotEnabledError** (p. 1869).

See also

dds::pub::DataWriter::register_instance (p. 1158)

dds::pub::DataWriter::unregister_instance (p. 1160)

dds::pub::DataWriter::key_value (p. 1164)

8.67.4.18 dispose_instance() [1/3]

```
template<typename T >
DataWriter &  dds::pub::DataWriter< T >::dispose_instance (
              const  dds::core::InstanceHandle & handle )  [inline]
```

Requests the middleware to delete the instance identified by the instance handle.

This operation is useful only for keyed data types. Using it for non-keyed types has no effect and reports no error.

When an instance is disposed, the **dds::pub::DataWriter** (p. 1138) communicates this state change to **dds::sub::DataReader** (p. 976) objects by propagating a dispose sample. When the instance changes to a disposed state, you

can see the state change on the DataReader by looking at **dds::sub::status::DataState::instance_state()** (p. 1123) in **dds::sub::SampleInfo::state()** (p. 2289). Disposed instances have the value **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617).

The resources allocated to dispose instances on the **DataWriter** (p. 1138) are not removed by default. The removal of the resources allocated to a dispose instance on the **DataWriter** (p. 1138) queue can be controlled by using the QoS **dds::core::policy::WriterDataLifecycle::autopurge_disposed_instances_delay** (p. 2664).

Likewise, on the DataReader, the removal of the resources associated with an instance in the dispose state can be controlled by using the QoS **dds::core::policy::ReaderDataLifecycle::autopurge_disposed_instances_delay** (p. 2142).

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is made available to **dds::sub::DataReader** (p. 976) objects by means of the source_timestamp attribute inside the **dds::sub::SampleInfo** (p. 2285).

The constraints on the values of the handle parameter and the corresponding error behavior are the same specified for the **dds::pub::DataWriter::unregister_instance** (p. 1160) operation.

This operation may block and time out (**dds::core::TimeoutError** (p. 2473)) under the same circumstances described for **dds::pub::DataWriter::write()** (p. 1148).

If there are no instance resources left, this operation may fail with **dds::core::OutOfResourcesError** (p. 1898). Calling **dds::pub::DataWriter::unregister_instance** (p. 1160) may help free up some resources.

Parameters

handle	The handle returned by a previous call to **register_instance()** (p. 1158) or **lookup_instance()** (p. 1165). If `instance_handle` doesn't correspond to an instance previously written or registered with this writer, this function fails with **dds::core::InvalidArgumentError** (p. 1619).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::TimeoutError** (p. 2473), **dds::core::OutOfResourcesError** (p. 1898) or **dds::core::NotEnabledError** (p. 1869).

See also

dds::pub::DataWriter::dispose_instance(const dds::core::InstanceHandle&,const dds::core::Time&) (p. 1163)

Relationship between registration, liveliness and ownership (p. 1902)

8.67.4.19 dispose_instance() [2/3]

```
template<typename T >
DataWriter &  dds::pub::DataWriter< T >::dispose_instance (
             const  dds::core::InstanceHandle & the_instance_handle,
             const  dds::core::Time & source_timestamp )  [inline]
```

Dispose an instance with a timestamp.

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **dds::pub::DataWriter::dispose_instance()** (p. 1162) operation.

This operation may block and time out (**dds::core::TimeoutError** (p. 2473)) under the same circumtances described for **dds::pub::DataWriter::write()** (p. 1148).

If there are no instance resources left, this operation may fail with **dds::core::OutOfResourcesError** (p. 1898). Calling **dds::pub::DataWriter::unregister_instance** (p. 1160) may help freeing up some resources.

Parameters

the_instance_handle	<<***in***>> *(p. 355)* Either the handle returned by a previous call to **dds::pub::DataWriter::register_instance** (p. 1158), or else the special value **dds::core::InstanceHandle::nil()** (p. 1614). If `T` has a key, `handle` must represent a registered instance of type `T`. Otherwise, this method may fail with **dds::core::InvalidArgumentError** (p. 1619).
source_timestamp	<<***in***>> *(p. 355)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. This timestamp will be available to the **dds::sub::DataReader** (p. 976) objects by means of the `source_timestamp` attribute inside the **dds::sub::SampleInfo** (p. 2285).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::TimeoutError** (p. 2473), **dds::core::OutOfResourcesError** (p. 1898) or **dds::core::NotEnabledError** (p. 1869).

See also

dds::pub::DataWriter::dispose_instance() (p. 1162)

8.67.4.20 key_value() [1/2]

```
template<typename T >
T &  dds::pub::DataWriter< T >::key_value (
             T & key_holder,
             const  dds::core::InstanceHandle & handle )  [inline]
```

Retrieve the instance key that corresponds to an instance handle.

Useful for keyed data types.

The operation will only fill the fields that form the `key` inside the `key_holder` instance. If the type has no keys, this method has no effect and exits with no error.

For keyed data types, this operation may fail with **dds::core::InvalidArgumentError** (p. 1619) if the `handle` does not correspond to an existing data-object known to the **dds::pub::DataWriter** (p. 1138).

Parameters

key_holder	<<***inout***>> *(p. 355)* a user data type specific key holder, whose `key` fields are filled by this operation. If `T` has no key, this method has no effect.

Parameters

handle	<<***in***>> *(p. 355)* the `instance` whose key is to be retrieved. If `T` *has* a key, `handle` must represent a registered instance of type T. Otherwise, this method will fail with **dds::core::InvalidArgumentError** (p. 1619). If `T` *has* a key and `handle` is **dds::core::InstanceHandle::nil()** (p. 1614), this method will fail with **dds::core::InvalidArgumentError** (p. 1619).

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869).

Returns

A reference to `key_holder`

See also

dds::sub::DataReader::key_value (p. 1003)

8.67.4.21 key_value() [2/2]

```
template<typename T >
dds::topic::TopicInstance< T > &  dds::pub::DataWriter< T >::key_value (
              dds::topic::TopicInstance< T > & key_holder,
             const  dds::core::InstanceHandle & handle )  [inline]
```

Retrieve the instance key that corresponds to an instance handle.

See also

key_value(T&, const dds::core::InstanceHandle&) (p. 1164)

8.67.4.22 lookup_instance()

```
template<typename T >
dds::core::InstanceHandle  dds::pub::DataWriter< T >::lookup_instance (
            const T & key_holder )  [inline]
```

Retrieve the instance handle that corresponds to an instance key_holder.

Useful for keyed data types.

This operation takes as a parameter an instance and returns a handle that can be used in subsequent operations that accept an instance handle as an argument. The instance parameter is only used for the purpose of examining the fields that define the key. This operation does not register the instance in question. If the instance has not been previously registered, or if for any other reason RTI Connext is unable to provide an instance handle, RTI Connext will return the special value HANDLE_NIL.

Parameters

key_holder	<<***in***>> *(*p. *355)* a user data type specific key holder.

Returns

the instance handle associated with this instance. If T has no key, this method has no effect and returns **dds::core::InstanceHandle::nil()** (p. 1614)

8.67.4.23 qos() [1/2]

```
template<typename T >
const  dds::pub::qos::DataWriterQos  dds::pub::DataWriter< T >::qos ( ) const  [inline]
```

Gets the DataWriterQos.

8.67.4.24 qos() [2/2]

```
template<typename T >
void  dds::pub::DataWriter< T >::qos (
            const  dds::pub::qos::DataWriterQos & the_qos )  [inline]
```

Sets the DataWriterQos.

This operation modifies the QoS of the **dds::pub::DataWriter** (p. 1138).

The **dds::core::policy::UserData** (p. 2591), **dds::core::policy::Deadline** (p. 1255), **dds::core::policy::LatencyBudget** (p. 1631), **dds::core::policy::OwnershipStrength** (p. 1906), **dds::core::policy::TransportPriority** (p. 2552), **dds::core::policy::Lifespan** (p. 1635) and **dds::core::policy::WriterDataLifecycle** (p. 2660) can be changed. The other policies are immutable.

Parameters

the_qos	<<***in***>> *(*p. *355)* The **dds::pub::qos::DataWriterQos** (p. 1226) to be set to. Policies must be consistent. Immutable policies cannot be changed after **dds::pub::DataWriter** (p. 1138) is enabled. The special value DATAWRITER_QOS_DEFAULT can be used to indicate that the QoS of the **dds::pub::DataWriter** (p. 1138) should be changed to match the current default **dds::pub::qos::DataWriterQos** (p. 1226) set in the **dds::pub::Publisher** (p. 1993).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::ImmutablePolicyError** (p. 1609) or **dds::core::InconsistentPolicyError** (p. 1610)

See also

dds::pub::qos::DataWriterQos (p. 1226) for rules on consistency among QoS

set_qos (abstract) (p. 1522)

Operations Allowed in Listener Callbacks (p. 1639)

8.67.4.25 operator<<() [4/4]

```
template<typename T >
DataWriter &  dds::pub::DataWriter< T >::operator<< (
            const  dds::pub::qos::DataWriterQos & the_qos )  [inline]
```

Set the DataWriterQos.

Parameters

the_qos	the new qos for this **DataWriter** (p. 1138).

See also

qos() (p. 1166)

8.67.4.26 operator>>()

```
template<typename T >
const  DataWriter &  dds::pub::DataWriter< T >::operator>> (
             dds::pub::qos::DataWriterQos & the_qos ) const  [inline]
```

Get the DataWriterQos.

Parameters

the_qos	the object to populate with the qos for this **DataWriter** (p. 1138).

See also

qos() (p. 1166)

8.67.4.27 topic()

```
template<typename T >
const  dds::topic::Topic< T > &  dds::pub::DataWriter< T >::topic ( ) const  [inline]
```

Get the Topic associated with this **DataWriter** (p. 1138).

8.67.4.28 publisher()

```
template<typename T >
const  dds::pub::Publisher &  dds::pub::DataWriter< T >::publisher ( ) const  [inline]
```

Get the **Publisher** (p. 1993) that owns this **DataWriter** (p. 1138).

8.67.4.29 wait_for_acknowledgments()

```
template<typename T >
void  dds::pub::DataWriter< T >::wait_for_acknowledgments (
            const  dds::core::Duration & max_wait )  [inline]
```

Blocks the calling thread until all data written by reliable **DataWriter** (p. 1138) entity is acknowledged, or until timeout expires.

This operation blocks the calling thread until either all data written by the reliable **dds::pub::DataWriter** (p. 1138) entity is acknowledged by (a) all reliable **dds::sub::DataReader** (p. 976) entities that are matched and alive and (b) by all required subscriptions, or until the duration specified by the `max_wait` parameter elapses, whichever happens first. A successful completion indicates that all the samples written have been acknowledged by all reliable matched data readers and by all required subscriptions; a timeout indicates that max_wait elapsed before all the data was acknowledged.

Note that if a thread is blocked in the call to wait_for_acknowledgments on a **DataWriter** (p. 1138) and a different thread writes new samples on the same **DataWriter** (p. 1138), the new samples must be acknowledged before unblocking the thread waiting on wait_for_acknowledgments.

If the **dds::pub::DataWriter** (p. 1138) does not have **dds::core::policy::Reliability** (p. 2149) kind set to RELIABLE, this operation will succeed immediately.

Parameters

max_wait	<<***in***>> *(p. 355)* Specifies maximum time to wait for acknowledgements **dds::core::Duration** (p. 1448) .

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::NotEnabledError** (p. 1869), **dds::core::TimeoutError** (p. 2473)

8.67.4.30 listener() [1/2]

```
template<typename T >
void  dds::pub::DataWriter< T >::listener (
             DataWriterListener< T > * l,
            const  dds::core::status::StatusMask & mask )  [inline]
```

Sets the **DataWriter** (p. 1138) listener.

[DEPRECATED] The use of **set_listener()** (p. 1169) is recommended. Unlike this function, set_listener receives a shared_ptr which simplifies the management of listener's lifecycle.

Parameters

l	The **DataWriterListener** (p. 1202) to set
mask	The **dds::core::status::StatusMask** (p. 2375) associated with the listener

8.67.4.31 listener() [2/2]

```
template<typename T >
DataWriterListener< T > *  dds::pub::DataWriter< T >::listener ( ) const  [inline]
```

Returns the listener currently associated with this **DataWriter** (p. 1138).

[DEPRECATED] Prefer **get_listener()** (p. 1170) instead of this function.

If there's no listener it returns NULL.

8.67.4.32 set_listener() [1/2]

```
template<typename T >
void  dds::pub::DataWriter< T >::set_listener (
            std::shared_ptr<  Listener > the_listener,
            const  dds::core::status::StatusMask & event_mask )  [inline]
```

Sets the listener associated with this writer.

The writer will hold a `shared_ptr` to the listener argument, ensuring that it is not deleted at least until this writer is deleted or the listener is reset with `set_listener(nullptr)`.

Warning

It's recommended that the listener implementation doesn't hold a permanent reference to the writer. If it does, the application needs to manually reset the listener or manually close this writer to ensure that there is no cycle that prevents the destruction of these two objects.

Parameters

the_listener	A shared pointer to the listener to receive status updates or `nullptr` to reset the current listener and stop receiving status updates.
event_mask	A mask that indicates which status updates will be notified to the listener

8.67.4.33 set_listener() [2/2]

```
template<typename T >
void  dds::pub::DataWriter< T >::set_listener (
            std::shared_ptr<  Listener > the_listener )  [inline]
```

Sets the listener associated with this writer.

If `the_listener` is not `nullptr`, this overload is equivalent to:
`writer.set_listener(the_listener, dds::core::status::StatusMask::all());`

If `the_listener` is `nullptr`, it is equivalent to:
`writer.set_listener(nullptr, dds::core::status::StatusMask::none());`

Parameters

the_listener	A shared pointer to the listener to receive status updates or `nullptr` to reset the current listener and stop receiving status updates.

8.67.4.34 get_listener()

```
template<typename T >
std::shared_ptr< Listener > dds::pub::DataWriter< T >::get_listener ( ) const [inline]
```

Returns the listener currently associated with this writer.

Returns

The shared pointer to the current listener or `nullptr` if there is currently no listener associated to this entity.

8.67.4.35 liveliness_lost_status()

```
template<typename T >
const dds::core::status::LivelinessLostStatus dds::pub::DataWriter< T >::liveliness_lost_status
( ) [inline]
```

Get the LivelinessLostStatus.

This also resets the status so that it is no longer considered changed.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.67.4.36 offered_deadline_missed_status()

```
template<typename T >
const dds::core::status::OfferedDeadlineMissedStatus dds::pub::DataWriter< T >::offered_↩
deadline_missed_status ( ) [inline]
```

Get the OfferedDeadlineMissedStatus.

This also resets the status so that it is no longer considered changed.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.67.4.37 offered_incompatible_qos_status()

```
template<typename T >
const dds::core::status::OfferedIncompatibleQosStatus dds::pub::DataWriter< T >::offered_↩
incompatible_qos_status ( ) [inline]
```

Get the OfferedIncompatibleQosStatus.

This also resets the status so that it is no longer considered changed.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.67.4.38 publication_matched_status()

```
template<typename T >
const dds::core::status::PublicationMatchedStatus dds::pub::DataWriter< T >::publication_↩
matched_status ( ) [inline]
```

Get the PublicationMatchedStatus.

This also resets the status so that it is no longer considered changed.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.67.4.39 assert_liveliness()

```
template<typename T >
void dds::pub::DataWriter< T >::assert_liveliness ( ) [inline]
```

Manually asserts the liveliness of this **DataWriter** (p. 1138).

This is used in combination with the **LIVELINESS** (p. 535) policy to indicate to RTI Connext that the **dds::pub::Data↩ Writer** (p. 1138) remains active.

You only need to use this operation if the **LIVELINESS** (p. 535) setting is either **dds::core::policy::Liveliness↩ Kind_def::MANUAL_BY_PARTICIPANT** (p. 1655) or **dds::core::policy::LivelinessKind_def::MANUAL_BY_TOPIC** (p. 1655). Otherwise, it has no effect.

Note: writing data via the **dds::pub::DataWriter::write()** (p. 1148) or **dds::pub::DataWriter::write(const T&,const dds::core::Time&)** (p. 1149) operation asserts liveliness on the **dds::pub::DataWriter** (p. 1138) itself, and its **dds↩ ::domain::DomainParticipant** (p. 1326). Consequently the use of **assert_liveliness()** (p. 1172) is only needed if the application is not writing data regularly.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869)

See also

dds::core::policy::Liveliness (p. 1645)

8.67.4.40 unregister_instance() [3/3]

```
template<typename T >
void unregister_instance (
              rti::pub::WriteParams & params )
```

<<***extension***>> *(p. 354)* Unregister an instance with parameters

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

See also

dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&) (p. 1179)

dds::pub::DataWriter::dispose_instance(rti::pub::WriteParams&) (p. 1173)

8.67.4.41 dispose_instance() [3/3]

```
template<typename T >
void dispose_instance (
              rti::pub::WriteParams & params )
```

<<***extension***>> *(p. 354)* Dispose an instance with parameters

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

See also

dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&) (p. 1179)

8.67.4.42 is_sample_app_acknowledged()

```
template<typename T >
bool is_sample_app_acknowledged (
            const  rti::core::SampleIdentity & sample_id )
```

<<***extension***>> *(p. 354)* Indicates if a sample is considered application-acknowledged

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This method can be used to see if a sample has been application acknowledged by all the matching DataReaders that were alive when the sample was written.

Parameters

sample_id	<<***in***>> *(p. 355)* Cannot be NULL. Sample identity.

Returns

true when the sample has been application acknowledged. Otherwise, false.

Exceptions

One	of the **Standard Exceptions** (p. 428)

Note

If the **DataWriter** (p. 1138) does not match with any DataReader that enables application acknowledgments (that is, a DataReader with **dds::core::policy::Reliability::acknowledgment_kind** (p. 2154) set to a value other than **rti::core::policy::AcknowledgmentKind_def::PROTOCOL** (p. 797)), this method returns false.

8.67.4.43 wait_for_asynchronous_publishing()

```
template<typename T >
void wait_for_asynchronous_publishing (
            const  dds::core::Duration & max_wait )
```

<<***extension***>> *(p. 354)* Blocks the calling thread until asynchronous sending is complete.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This operation blocks the calling thread (up to `max_wait`) until all data written by the asynchronous **dds::pub::DataWriter** (p. 1138) is sent and acknowledged (if reliable) by all matched **dds::sub::DataReader** (p. 976) entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that `max_wait` elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort **dds::sub::DataReader** (p. 976) is complete in addition to what **dds::pub::DataWriter::wait_for_acknowledgments** (p. 1168) provides.

If the **dds::pub::DataWriter** (p. 1138) does not have **rti::core::policy::PublishMode** (p. 2013) kind set to **rti::core::policy::PublishModeKind_def::ASYNCHRONOUS** (p. 2019) the operation will complete immediately .

Parameters

max_wait	<<***in***>> *(p. 355)* Specifies maximum time to wait for acknowledgements.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::NotEnabledError** (p. 1869), **dds::core::TimeoutError** (p. 2473)

8.67.4.44 reliable_writer_cache_changed_status()

```
template<typename T >
rti::core::status::ReliableWriterCacheChangedStatus reliable_writer_cache_changed_status ( )
```

<<***extension***>> *(p. 354)* Get the reliable cache status for this writer.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This also resets the status so that it is no longer considered changed.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.67.4.45 reliable_reader_activity_changed_status()

```
template<typename T >
rti::core::status::ReliableReaderActivityChangedStatus reliable_reader_activity_changed_status ( )
```

<<***extension***>> *(p. 354)* Get the reliable reader activity changed status for this writer

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This also resets the status so that it is no longer considered changed.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.67.4.46 datawriter_cache_status()

```
template<typename T >
rti::core::status::DataWriterCacheStatus datawriter_cache_status ( )
```

<<***extension***>> *(p. 354)* Get the cache status for this writer

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This also resets the status so that it is no longer considered changed.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869).

8.67.4.47 datawriter_protocol_status()

```
template<typename T >
rti::core::status::DataWriterProtocolStatus datawriter_protocol_status ( )
```

<<***extension***>> *(p. 354)* Get the protocol status for this writer

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This also resets the status so that it is no longer considered changed.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869).

8.67.4.48 matched_subscription_datawriter_protocol_status() [1/2]

```
template<typename T >
rti::core::status::DataWriterProtocolStatus matched_subscription_datawriter_protocol_status (
            const  dds::core::InstanceHandle & subscription_handle )
```

*<<**extension**>> (p. 354)* Get the datawriter protocol status for this writer, per matched subscription identified by the subscription_handle.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This also resets the status so that it is no longer considered changed.

Note: Status for a remote entity is only kept while the entity is alive. Once a remote entity is no longer alive, its status is deleted.

Parameters

subscription_handle	*<<**in**>> (p. 355)* Handle to a specific subscription associated with the **dds::sub::DataReader** (p. 976). Must correspond to a subscription currently associated with the **dds::pub::DataWriter** (p. 1138).

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869).

8.67.4.49 matched_subscription_datawriter_protocol_status() [2/2]

```
template<typename T >
rti::core::status::DataWriterProtocolStatus matched_subscription_datawriter_protocol_status (
            const  rti::core::Locator & subscription_locator )
```

<<***extension***>> *(p. 354)* Get the datawriter protocol status for this writer, per matched subscription identified by the locator

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This also resets the status so that it is no longer considered changed.

Note: Status for a remote entity is only kept while the entity is alive. Once a remote entity is no longer alive, its status is deleted.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869).

Parameters

subscription_locator	<<***in***>> *(p. 355)* Locator to a specific locator associated with the DataReader. Must correspond to a locator of one or more subscriptions currently associated with the **DataWriter** (p. 1138).

8.67.4.50 service_request_accepted_status()

```
template<typename T >
rti::core::status::ServiceRequestAcceptedStatus service_request_accepted_status ( )
```

<<***extension***>> *(p. 354)* Get the service request accepted status for this writer

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This also resets the status so that it is no longer considered changed.

Note: Status for a remote entity is only kept while the entity is alive. Once a remote entity is no longer alive, its status is deleted.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869).

8.67.4.51 flush()

```
template<typename T >
void flush ( )
```

<<***extension***>> *(*p. *354)* Flushes the batch in progress in the context of the calling thread.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

After being flushed, the batch is available to be sent on the network.

If the **dds::pub::DataWriter** (p. 1138) does not have **rti::core::policy::PublishMode** (p. 2013) kind set to **rti::core::policy::PublishModeKind_def::ASYNCHRONOUS** (p. 2019), the batch will be sent on the network immediately (in the context of the calling thread).

If the **dds::pub::DataWriter** (p. 1138) does have **rti::core::policy::PublishMode** (p. 2013) kind set to **rti::core::policy::PublishModeKind_def::ASYNCHRONOUS** (p. 2019), the batch will be sent in the context of the asynchronous publishing thread.

This operation may block in the same conditions as **dds::pub::DataWriter::write()** (p. 1148).

If this operation does block, the RELIABILITY max_blocking_time configures the maximum time the write operation may block (waiting for space to become available). If max_blocking_time elapses before the DDS_DataWriter is able to store the modification without exceeding the limits, the operation will fail with DDS_RETCODE_TIMEOUT.

MT Safety:

flush() (p. 1178) is only thread-safe with batching if **rti::core::policy::Batch::thread_safe_write** (p. 886) is TRUE.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::TimeoutError** (p. 2473), **dds::core::OutOfResourcesError** (p. 1898) or **dds::core::NotEnabledError** (p. 1869).

8.67.4.52 write() [11/11]

```
template<typename T >
void write (
            const T & instance_data,
             rti::pub::WriteParams & params )
```

<<***extension***>> *(*p. *354)* Write with advanced parameters

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Allows provision of the sample identity, related sample identity, source timestamp, instance handle, and publication priority contained in `params`.

This operation may block and time out (**dds::core::TimeoutError** (p. 2473)) under the same circumstances described for **dds::pub::DataWriter::write()** (p. 1148).

If there are no instance resources left, this operation may fail with **dds::core::OutOfResourcesError** (p. 1898). Calling **dds::pub::DataWriter::unregister_instance(rti::pub::WriteParams&)** (p. 1173) may help free up some resources.

This operation may fail with **dds::core::InvalidArgumentError** (p. 1619) under the same circumstances described for the **dds::pub::DataWriter::write()** (p. 1148).

Parameters

instance_data	<<***in***>> *(p. 355)* The data to write.
params	<<***inout***>> *(p. 355)* The write parameters. Note that this is an inout parameter if you activate **rti::pub::WriteParams::replace_automatic_values** (p. 2644); otherwise it won't be modified.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::TimeoutError** (p. 2473), **dds::core::OutOfResourcesError** (p. 1898) or **dds::core::NotEnabledError** (p. 1869).

See also

dds::pub::DataWriter::write() (p. 1148)

dds::sub::DataReader (p. 976)

8.67.4.53 write_noexcept() [1/5]

```
template<typename T >
rti::core::Result< void > write_noexcept (
             const T & sample )  [noexcept]
```

<<***extension***>> *(p. 354)* Alternative to **write()** (p. 1148) that doesn't throw exceptions.

This operation behaves exactly like **write(const T&)** (p. 1148) but is guaranteed to not throw exceptions.

As an extension function, it must be accessed with the `extensions()` method or with the `->` operator. Since `extensions()` can throw if the object is not a valid reference, the `->` operator is recommended.

Example:

```
auto result = writer->write_noexcept(sample);
if (!result.is_ok()) {
    if (result.get_return_code() == DDS_RETCODE_TIMEOUT) {
       // Handle timeout
    }
    // ...
}
```

Returns

The Result of the operation.

8.67.4.54 write_noexcept() [2/5]

```
template<typename T >
rti::core::Result< void > write_noexcept (
            const T & sample,
            const  dds::core::Time & timestamp )  [noexcept]
```

<<***extension***>> *(*p. *354)* Alternative to **write()** (p. 1148) that doesn't throw exceptions.

This operation behaves exactly like **write(const T&, const dds::core::Time&)** (p. 1149) but is guaranteed to not throw exceptions.

As an extension function, it must be accessed with the `extensions()` method or with the `->` operator. Since `extensions()` can throw if the object is not a valid reference, the `->` operator is recommended.

Returns

The Result of the operation.

8.67.4.55 write_noexcept() [3/5]

```
template<typename T >
rti::core::Result< void > write_noexcept (
            const T & sample,
            const  dds::core::InstanceHandle & instance )  [noexcept]
```

<<***extension***>> *(*p. *354)* Alternative to **write()** (p. 1148) that doesn't throw exceptions.

This operation behaves exactly like **write(const T&, const dds::core::InstanceHandle&)** (p. 1149) but is guaranteed to not throw exceptions.

As an extension function, it must be accessed with the `extensions()` method or with the `->` operator. Since `extensions()` can throw if the object is not a valid reference, the `->` operator is recommended.

Returns

The Result of the operation.

8.67.4.56 write_noexcept() [4/5]

```
template<typename T >
rti::core::Result< void > write_noexcept (
            const T & sample,
            const  dds::core::InstanceHandle & instance,
            const  dds::core::Time & timestamp )  [noexcept]
```

<<***extension***>> *(p. 354)* Alternative to **write()** (p. 1148) that doesn't throw exceptions.

This operation behaves exactly like **write(const T&, const dds::core::InstanceHandle&, const dds::core::Time&)** (p. 1153) but is guaranteed to not throw exceptions.

As an extension function, it must be accessed with the `extensions()` method or with the `->` operator. Since `extensions()` can throw if the object is not a valid reference, the `->` operator is recommended.

Returns

The Result of the operation.

8.67.4.57 write_noexcept() [5/5]

```
template<typename T >
rti::core::Result< void > write_noexcept (
            const T & instance_data,
             rti::pub::WriteParams & params )  [noexcept]
```

<<***extension***>> *(p. 354)* Alternative to **write()** (p. 1148) that doesn't throw exceptions.

This operation behaves exactly like **write(const T&, rti::pub::WriteParams&)** (p. 1179) but is guaranteed to not throw exceptions.

As an extension function, it must be accessed with the `extensions()` method or with the `->` operator. Since `extensions()` can throw if the object is not a valid reference, the `->` operator is recommended.

Returns

The Result of the operation.

8.67.4.58 create_data()

```
template<typename T >
T * create_data ( )
```

<<***extension***>> *(p. 354)* Create data from this writer and initialize it

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Returns

Newly created data type

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

dds::pub::DataWriter::delete_data (p. 1183)

8.67.4.59 delete_data()

```
template<typename T >
bool delete_data (
             T * sample )
```

<<***extension***>> *(p. 354)* Delete data created with **create_data()** (p. 1182).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The behavior of this API is identical to FooTypeSupport::delete_data.

Parameters

sample	<<***in***>> *(p. 355)* Cannot be NULL.

Returns

true on success.

See also

dds::pub::DataWriter::create_data (p. 1182)

8.67.4.60 get_loan()

```
template<typename T >
T * get_loan ( )
```

<<***extension***>> *(p. 354)* Get a loaned sample from this writer

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This operation is supported while using **Zero Copy transfer** (p. 50) over shared memory" or \ref RTIFlatDataModule "FlatData language binding". For FlatData types, this function should be used directly if the type is final; if the type is mutable, **rti::flat::build_data()** (p. 411) should be used to obtain a sample builder to work with the loaned sample.

The loaned sample is obtained from a DataWriter-managed sample pool and is uninitialized by default. An initialized sample can be obtained by setting **rti::core::policy::DataWriterResourceLimits::initialize_writer_loaned_sample** (p. 1248) to true. The **rti::core::policy::DataWriterResourceLimits::writer_loaned_sample_allocation** (p. 1247) settings can be used to configure the DataWriter-managed sample pool.

dds::pub::DataWriter::get_loan (p. 1183) fails with **dds::core::OutOfResourcesError** (p. 1898) if **rti::core::↩ AllocationSettings::max_count** (p. 804) samples have been loaned, and none of those samples has been written with **dds::pub::DataWriter::write()** (p. 1148) or discarded via **dds::pub::DataWriter::discard_loan** (p. 1185).

Samples returned from **dds::pub::DataWriter::get_loan** (p. 1183) have an associated state. Due to the optimized nature of the write operation while using Zero Copy transfer over shared memory or FlatData language binding, this sample state is used to control when a sample is available for reuse after the write operation. The possible sample states are free, allocated, removed or serialized. A sample that has never been allocated is "free". **dds::pub::Data↩ Writer::get_loan** (p. 1183) takes a "free" or "removed" sample and makes it "allocated". When a sample is written, its state transitions from "allocated" to "serialized", and the **DataWriter** (p. 1138) takes responsibility for returning the sample back to its sample pool. The sample remains in the "serialized" state until it is removed from the **DataWriter** (p. 1138) queue. For a reliable **DataWriter** (p. 1138), the sample is removed from the **DataWriter** (p. 1138)'s queue when the sample is acknowledged by all DataReaders. For a best-effort **DataWriter** (p. 1138), the sample is removed from the queue immediately after the write operation. After the sample is removed from the **DataWriter** (p. 1138) queue, the sample is put back into the sample pool, and its state transitions from "serialized" to "removed". At this time, a new call to **dds::pub::DataWriter::get_loan** (p. 1183) may return the same sample.

A loaned sample should not be reused to write a new value after the first write operation. Instead, a new sample from **dds::pub::DataWriter::get_loan** (p. 1183) should be used to write the new value. A loaned sample that has not been written can be returned to the **DataWriter** (p. 1138)'s sample pool by using **dds::pub::DataWriter::discard_loan** (p. 1185). If the write operation fails, then the sample can be used again with a write or discard_loan operation. Disposing or unregistering an instance with loaned samples follows the same pattern. A loaned sample used successfully with a dispose or unregister operation cannot be used again. But if the dispose or unregister operation fails, the sample is available for reuse.

A **DataWriter** (p. 1138) cannot write managed samples (created with get_loan) and unmanaged samples (created in any other way) at the same time. The first call to get_loan automatically prepares this **DataWriter** (p. 1138) to work with managed samples. Calls to get_loan will fail with **dds::core::PreconditionNotMetError** (p. 1941) if an unmanaged sample was written with this **DataWriter** (p. 1138) earlier. Similarly, **dds::pub::DataWriter::write()** (p. 1148) will fail to write an unmanaged sample if get_loan was called.

Returns

a loaned sample

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::OutOfResourcesError** (p. 1898) or **dds::core::PreconditionNotMetError** (p. 1941).

See also

dds::pub::DataWriter::discard_loan (p. 1185)

Referenced by **rti::flat::build_data()**.

8.67.4.61 discard_loan()

```
template<typename T >
void discard_loan (
            T & sample )
```

<<***extension***>> *(p. 354)* Discard a loaned sample from this writer

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This operation is supported while using **Zero Copy transfer** (p. 50) over shared memory" or the **FlatData language binding** (p. 419).

A loaned sample that hasn't been written can be returned to the **DataWriter** (p. 1138) with this operation.

Parameters

sample	<<***in***>> *(p. 355)* loaned sample to be discarded.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869).

See also

dds::pub::DataWriter::get_loan (p. 1183)

Referenced by **rti::flat::discard_builder()**.

8.67.4.62 register_instance() [3/3]

```
template<typename T >
const  dds::core::InstanceHandle register_instance (
            const T & key,
             rti::pub::WriteParams & params )
```

<<***extension***>> *(p. 354)* Registers and instance with parameters

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

See also

dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&) (p. 1179)

8.67.5 Friends And Related Function Documentation

8.67.5.1 ignore() [1/2]

```
template<typename T >
void ignore (
              dds::domain::DomainParticipant & participant,
             const  dds::core::InstanceHandle & handle )  [related]
```

Instructs RTI Connext to locally ignore a publication.

A publication is defined by the association of a topic name, user data, and partition set on the **dds::pub::Publisher** (p. 1993) (see **dds::topic::PublicationBuiltinTopicData** (p. 1977)). After this call, any data written by that publication's **dds::pub::DataWriter** (p. 1138) will be ignored.

This operation can be used to ignore local *and* remote DataWriters.

The publication (**DataWriter** (p. 1138)) to ignore is identified by the `handle` argument.

- To ignore a *remote* **DataWriter** (p. 1138), the `handle` can be obtained from the **dds::sub::SampleInfo** (p. 2285) retrieved when reading data samples from the built-in **dds::sub::DataReader** (p. 976) for the publication topic.
- To ignore a *local* **DataWriter** (p. 1138), the `handle` can be obtained by calling **dds::core::Entity::instance_handle()** (p. 1525) for the local **DataWriter** (p. 1138).

There is no way to reverse this operation.

Parameters

participant	The DomainParticipant where the publication will be ignored
handle	<<***in***>> *(p. 355)* Handle of the **dds::pub::DataWriter** (p. 1138) to be ignored.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::OutOfResourcesError** (p. 1898) or **dds::core::NotEnabledError** (p. 1869)

See also

dds::topic::PublicationBuiltinTopicData (p. 1977)

dds::topic::publication_topic_name() (p. 443)

dds::sub::builtin_subscriber (p. 671)

8.67.5.2 ignore() [2/2]

```
template<typename FwdIterator >
void ignore (
              dds::domain::DomainParticipant & participant,
             FwdIterator begin,
             FwdIterator end )  [related]
```

Instructs RTI Connext to locally ignore several publications.

Template Parameters

FwdIterator	A forward iterator whose value type is **dds::core::InstanceHandle** (p. 1612)

See also

ignore(dds::domain::DomainParticipant&, const dds::core::InstanceHandle&) (p. 1186);

References **dds::pub::ignore()**.

8.67.5.3 matched_subscriptions() [1/2]

```
template<typename T >
dds::core::InstanceHandleSeq matched_subscriptions (
             const  dds::pub::DataWriter< T > & writer )  [related]
```

Retrieve the list of subscriptions currently associated with a **dds::pub::DataWriter** (p. 1138).

A subscription is considered to be matching if all of the following criteria are true:

- The subscription is within the same domain as this publication.
- The subscription has a matching **dds::topic::Topic** (p. 2474).
- The subscription has compatible QoS.
- If the applications are using partitions, the subscription shares a common partition with this publication.

- The **dds::domain::DomainParticipant** (p. 1326) has not indicated that the subscription's **dds::domain::DomainParticipant** (p. 1326) should be "ignored" by means of the **dds::pub::ignore** (p. 647) API.
- If the publication is using the **rti::core::policy::MultiChannel** (p. 1749) and the subscription is using a **dds::topic::ContentFilteredTopic** (p. 954), there is an intersection between at least one of the associated filters.
- If the endpoints need to exchange key material to communicate (i.e., they are securing their communications), the writer has completed the key exchange with reader.

The handles returned list are the ones that RTI Connext uses to locally identify the corresponding matched **dds::sub::DataReader** (p. 976) entities. These handles match the ones that appear in the **dds::sub::SampleInfo::instance_handle** (p. 2290) field of the **dds::sub::SampleInfo** (p. 2285) when reading the **dds::topic::subscription_topic_name()** (p. 444) builtin topic.

This API may return the subscription handles of subscriptions that are inactive. **dds::pub::DataWriter::is_matched_subscription_active** (p. 1191) can be used to check this.

The maximum number of matches possible is configured with **rti::core::policy::DomainParticipantResourceLimits** (p. 1391). .

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::NotEnabledError** (p. 1869)

8.67.5.4 matched_subscriptions() [2/2]

```
template<typename T , typename FwdIterator >
FwdIterator matched_subscriptions (
            const  dds::pub::DataWriter< T > & writer,
            FwdIterator begin,
            FwdIterator end )  [related]
```

Retrieve the list of subscriptions currently associated with a **dds::pub::DataWriter** (p. 1138).

This operation is similar to **matched_subscriptions(const dds::pub::DataWriter<T>&)** (p. 1187) but it copies the instance handles into an iterator range.

Template Parameters

FwdIterator	A forward iterator whose value type is **dds::core::InstanceHandle** (p. 1612)

References **dds::pub::matched_subscriptions()**.

8.67.5.5 matched_subscription_data() [1/2]

```
template<typename T >
const  dds::topic::SubscriptionBuiltinTopicData matched_subscription_data (
             const  dds::pub::DataWriter< T > & writer,
             const  dds::core::InstanceHandle & subscription_handle )  [related]
```

Retrieves information on a subscription that is currently associated with a **dds::pub::DataWriter** (p. 1138).

The subscription_handle must correspond to a subscription currently associated with the **dds::pub::DataWriter** (p. 1138). Otherwise, the operation will fail and fail with **dds::core::InvalidArgumentError** (p. 1619). Use **dds::pub::matched_subscriptions()** (p. 648) to find the subscriptions that are currently matched with the **dds::pub::DataWriter** (p. 1138).

The above information is also available through **dds::sub::DataReaderListener::on_data_available()** (p. 1058) (if a reader listener is installed on the dds::sub::DataReader<dds::topic::SubscriptionBuiltinTopicData>).

When the subscription data is updated, for example when the content filter property changes, there is a small window of time in between when the **DataWriter** (p. 1138) is made aware of these changes and when they actually take effect. Taking effect in this example means that the **DataWriter** (p. 1138) will perform writer-side filtering using the new filter property values (filter expression and/or parameters).

When the **DataWriter** (p. 1138) is made aware of the changes they will first be seen in the **dds::sub::DataReaderListener::on_data_available()** (p. 1058) of the dds::sub::DataReader<dds::topic::SubscriptionBuiltinTopicData>. When these changes are applied, they will be seen in the output of this API because this API blocks until the most recent changes known to the **DataWriter** (p. 1138) have taken effect. This API will only block when called outside of a listener callback, in order to not block the internal threads from making progress.

If application behavior depends on being made aware of information about a subscription only after it has taken effect on the **DataWriter** (p. 1138), the recommended pattern for usage of this API is to wait for subscription data to be received either through polling this API or by installing a listener on the dds::sub::DataReader<dds::topic::SubscriptionBuiltinTopicData>. When a new sample is received by the builtin DataReader, this API may be called in a separate thread and will return the expected matched subscription data once it has been applied to the **DataWriter** (p. 1138).

Because this API blocks, it is possible for this API to time out while waiting for the changes to be applied. A timeout may happen if the DataReader's subscription data is changing rapidly, preventing the **DataWriter** (p. 1138) from returning valid information before newer data has been received, or if an application is performing a task in a listener callback, thereby preventing the middleware's threads from executing events in a timely manner.

Note: This operation does not retrieve the **dds::topic::SubscriptionBuiltinTopicData::type** (p. 2434). This information is available through **dds::sub::DataReaderListener::on_data_available()** (p. 1058) (if a reader listener is installed on the dds::sub::DataReader<dds::topic::SubscriptionBuiltinTopicData>).

Parameters

writer	The **DataWriter** (p. 1138) to which the subscription is associated
subscription_handle	<<***in***>> *(p. 355)*. Handle to a specific subscription associated with the **dds::sub::DataReader** (p. 976). Must correspond to a subscription currently associated with the **dds::pub::DataWriter** (p. 1138).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::NotEnabledError** (p. 1869), or **dds::core::TimeoutError** (p. 2473)

8.67.5.6 matched_subscription_participant_data()

```
template<typename T >
dds::topic::ParticipantBuiltinTopicData matched_subscription_participant_data (
            const  dds::pub::DataWriter< T > & writer,
            const  dds::core::InstanceHandle & handle )  [related]
```

<<***extension***>> *(p. 354)* This operation retrieves the information on the discovered **dds::domain::Domain↩ Participant** (p. 1326) associated with the subscription that is currently matching with the **dds::pub::DataWriter** (p. 1138).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

The `subscription_handle` must correspond to a subscription currently associated with the **dds::pub::DataWriter** (p. 1138). Otherwise, the operation will fail with **dds::core::InvalidArgumentError** (p. 1619). The operation may also fail with **dds::core::PreconditionNotMetError** (p. 1941) if the subscription corresponds to the same **dds::domain↩ ::DomainParticipant** (p. 1326) that the **DataWriter** (p. 1138) belongs to. Use **dds::pub::matched_subscriptions()** (p. 648) to find the subscriptions that are currently matched with the **dds::pub::DataWriter** (p. 1138).

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::NotEnabledError** (p. 1869)

Template Parameters

T	The topic-type that the **DataWriter** (p. 1138) subscribes to

Parameters

writer	The writer to lookup the matched participant data of
handle	The InstanceHandle to a specific subscription. Must correspond to a subscription currently associated with the **DataWriter** (p. 1138).

Returns

The **dds::topic::ParticipantBuiltinTopicData** (p. 1908) of the DomainParticipant of a matched subscription of a **dds::pub::DataWriter** (p. 1138)

8.67.5.7 matched_subscriptions_locators()

```
template<typename T >
rti::core::LocatorSeq matched_subscriptions_locators (
            const  dds::pub::DataWriter< T > & writer )  [related]
```

<<***extension***>> *(p. 354)* Retrieve the list of locators for subscriptions currently associated with this **DataWriter** (p. 1138).

The locators returned are the ones that are used by the DDS implementation to communicate with the corresponding matched DataReaders.

Template Parameters

T	The topic-type that the **DataWriter** (p. 1138) publishes.

Parameters

writer	The **DataWriter** (p. 1138).

Returns

The list of locators

8.67.5.8 is_matched_subscription_active()

```
template<typename T >
bool is_matched_subscription_active (
            const  dds::pub::DataWriter< T > & writer,
            const  dds::core::InstanceHandle & handle )  [related]
```

<<***extension***>> *(p. 354)* Check if a matched subscription is active.

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Template Parameters

T	The topic-type that the **DataWriter** (p. 1138) publishes.

Parameters

writer	The **DataWriter** (p. 1138).
handle	The **dds::core::InstanceHandle** (p. 1612) of the matched subscription.

This API is used for querying the endpoint liveliness of a matched subscription. A matched subscription will be marked as inactive when it becomes nonprogessing (e.g., not responding to a **DataWriter** (p. 1138)'s heartbeats, or letting its internal queue fill up without taking the available data). Note that if the participant associated with the matched subscription loses liveliness, the **dds::core::InstanceHandle** (p. 1612) will become invalid and this function will fail with **dds::core::InvalidArgumentError** (p. 1619).

Returns

A boolean indicating whether or not the matched subscription is active.

8.67.5.9 matched_subscription_data() [2/2]

```
template<typename T >
std::vector< dds::topic::SubscriptionBuiltinTopicData > matched_subscription_data (
            const dds::pub::DataWriter< T > & writer ) [related]
```

<<***extension***>> *(p. 354)* Obtain the SubscriptionBuiltinTopicData for all of the subscriptions matched with a **Data↩ Writer** (p. 1138).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

This API retrieves the matched publication data from all of the subscriptions currently matched a **DataWriter** (p. 1138).

Template Parameters

T	The topic-type that the **DataWriter** (p. 1138) publishes.

Parameters

writer	The **DataWriter** (p. 1138).

Returns

A std::vector containing all of the matched subscription data.

8.67.5.10 find() [1/2]

```
template<typename Writer , typename FwdIterator >
uint32_t find (
            const  dds::pub::Publisher & publisher,
            RTI_STRING_IN topic_name,
            FwdIterator begin,
            uint32_t max_size )  [related]
```

Retrieves the **dds::pub::DataWriter** (p. 1138) for a specific topic name.

This returned **dds::pub::DataWriter** (p. 1138) is either enabled or disabled.

If more than one **dds::pub::DataWriter** (p. 1138) is attached to the **dds::pub::Publisher** (p. 1993) with the same `topic_name`, then this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **dds::pub::DataWriter** (p. 1138) in one thread while another thread is simultaneously creating or destroying that **dds::pub::DataWriter** (p. 1138).

This function retrieves a previously-created **DataWriter** (p. 1138) belonging to the **dds::pub::Publisher** (p. 1993) that is attached to a **dds::topic::Topic** (p. 2474) with a matching topic name. If no such **DataWriter** (p. 1138) exists, the operation will return an empty container. The use of this operation on the built-in **Publisher** (p. 1993) allows access to the built-in **DataWriter** (p. 1138) entities for the built-in topics

Template Parameters

Writer	The type of the writer. It can be **dds::pub::AnyDataWriter** (p. 815), or an instantiation of dds::pub::DataWriter<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))
FwdIterator	The type of forward iterator passed to this function

Parameters

publisher	The **dds::pub::Publisher** (p. 1993) the **DataWriter** (p. 1138) belongs to
topic_name	Name of the **dds::topic::Topic** (p. 2474) associated with the **DataWriter** (p. 1138) that is to be looked up.
begin	A forward iterator to the position in the destination container to insert the found **DataWriter** (p. 1138) in
max_size	Only 1 **DataWriter** (p. 1138) will be returned from this function

Returns

The number of DataWriters that were found (either 0 or 1)

8.67.5.11 find() [2/2]

```
template<typename Writer , typename BinIterator >
uint32_t find (
            const  dds::pub::Publisher & publisher,
            RTI_STRING_IN topic_name,
            BinIterator begin )  [related]
```

Retrieves the **dds::pub::DataWriter** (p. 1138) for a specific topic name.

This returned **dds::pub::DataWriter** (p. 1138) is either enabled or disabled.

If more than one **dds::pub::DataWriter** (p. 1138) is attached to the **dds::pub::Publisher** (p. 1993) with the same `topic_name`, then this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **dds::pub::DataWriter** (p. 1138) in one thread while another thread is simultaneously creating or destroying that **dds::pub::DataWriter** (p. 1138).

This function retrieves a previously created **DataWriter** (p. 1138) belonging to the **dds::pub::Publisher** (p. 1993) that is attached to a **dds::topic::Topic** (p. 2474) with a matching topic name. If no such **DataWriter** (p. 1138) exists, the operation will return an empty container. The use of this operation on the built-in **Publisher** (p. 1993) allows access to the built-in **DataWriter** (p. 1138) entities for the built-in topics

Template Parameters

Writer	The type of the writer. It can be **dds::pub::AnyDataWriter** (p. 815), or an instantiation of dds::pub::DataWriter<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))
BinIterator	BinIterator The type of back-inserting iterator passed to this function

Parameters

publisher	The **dds::pub::Publisher** (p. 1993) the **DataWriter** (p. 1138) belongs to
topic_name	Name of the **dds::topic::Topic** (p. 2474) associated with the **DataWriter** (p. 1138) that is to be looked up.
begin	A back-inserting iterator to the position in the destination container to insert the found **DataWriter** (p. 1138) into

Returns

The number of DataWriters that were found (either 0 or 1)

8.67.5.12 find_datawriters() [1/2]

```
template<typename AnyDataWriterBackInsertIterator >
uint32_t find_datawriters (
              dds::pub::Publisher publisher,
             AnyDataWriterBackInsertIterator begin )  [related]
```

Retrieve all the **dds::pub::DataWriter** (p. 1138) created from this **dds::pub::Publisher** (p. 1993).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Template Parameters

AnyDataWriterBackInsertIterator	An iterator whose value_type is **dds::pub::AnyDataWriter** (p. 815)

Parameters

publisher	The **dds::pub::Publisher** (p. 1993) the DataWriters belong to
begin	A back-inserting iterator to the position in the destination container to insert the found DataWriters into

Returns

The number of found DataWriters

See also

Looking up DataWriters (p. 314)

8.67.5.13 find_datawriters() [2/2]

```
template<typename AnyDataWriterForwardIterator >
uint32_t find_datawriters (
              dds::pub::Publisher publisher,
             AnyDataWriterForwardIterator begin,
             uint32_t max_size )  [related]
```

Retrieve all the **dds::pub::DataWriter** (p. 1138) created from this **dds::pub::Publisher** (p. 1993).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Template Parameters

AnyDataWriterForwardIterator	An iterator whose `value_type` is **dds::pub::AnyDataWriter** (p. 815)

Parameters

publisher	The **dds::pub::Publisher** (p. 1993) the DataWriters belong to
begin	A forward iterator to the first position in the destination container to copy the found DataWriters into
max_size	The maximum number of DataWriters to return

Returns

The number of found DataWriters

See also

Looking up DataWriters (p. 314)

8.67.5.14 find_datawriter_by_topic_name()

```
template<typename Writer >
Writer find_datawriter_by_topic_name (
               dds::pub::Publisher publisher,
              const std::string & topic_name )  [related]
```

<<***extension***>> *(p. 354)* Retrieves a **dds::pub::DataWriter** (p. 1138) with the given name within the **dds::pub::Publisher** (p. 1993)

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Every **dds::pub::DataWriter** (p. 1138) in the system has an entity name which is configured and stored in the <<***extension***>> *(p. 354)* EntityName policy, **ENTITY_NAME** (p. 532).

This operation retrieves the **dds::pub::DataWriter** (p. 1138) within the **dds::pub::Publisher** (p. 1993) whose name matches the one specified. If there are several **dds::pub::DataWriter** (p. 1138) with the same name within the **dds::pub::Publisher** (p. 1993), the operation returns the first matching occurrence.

Template Parameters

Writer	The type of the writer. It can be **dds::pub::AnyDataWriter** (p. 815), or an instantiation of dds::pub::DataWriter<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))

Parameters

publisher	The **dds::pub::Publisher** (p. 1993) that created the **DataWriter** (p. 1138) to find
topic_name	Entity name of the **DataWriter** (p. 1138) to find

Returns

The **AnyDataWriter** (p. 815) with the given name

See also

Looking up DataWriters (p. 314)

References **dds::core::null**.

8.67.5.15 find_datawriter_by_name() [1/2]

```
template<typename Writer >
Writer find_datawriter_by_name (
              dds::pub::Publisher publisher,
             const std::string & datawriter_name )  [related]
```

<<***extension***>> *(p. 354)* Retrieves a **dds::pub::DataWriter** (p. 1138) with the given name within the **dds::pub::Publisher** (p. 1993)

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Every **dds::pub::DataWriter** (p. 1138) in the system has an entity name which is configured and stored in the <<***extension***>> *(p. 354)* EntityName policy, **ENTITY_NAME** (p. 532).

This operation retrieves the **dds::pub::DataWriter** (p. 1138) within the **dds::pub::Publisher** (p. 1993) whose name matches the one specified. If there are several **dds::pub::DataWriter** (p. 1138) with the same name within the **dds::pub::Publisher** (p. 1993), the operation returns the first matching occurrence.

Template Parameters

Writer	The type of the writer. It can be **dds::pub::AnyDataWriter** (p. 815), or an instantiation of dds::pub::DataWriter<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))

Parameters

publisher	The **dds::pub::Publisher** (p. 1993) that created the **DataWriter** (p. 1138) to find

Parameters

datawriter_name	Entity name of the **DataWriter** (p. 1138) to find

Returns

The WRITER with the given name

See also

Looking up DataWriters (p. 314)

References **dds::core::null**.

8.67.5.16 find_datawriter_by_name() [2/2]

```
template<typename Writer >
Writer find_datawriter_by_name (
              dds::domain::DomainParticipant participant,
             const std::string & datawriter_name )  [related]
```

<<***extension***>> *(p. 354)* Retrieves a **dds::pub::DataWriter** (p. 1138) within the **dds::domain::DomainParticipant** (p. 1326) with the given name

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Every **dds::pub::DataWriter** (p. 1138) in the system has an entity name which is configured and stored in the Entity↩ Name policy, **ENTITY_NAME** (p. 532).

Every **dds::pub::Publisher** (p. 1993) in the system has an entity name which is also configured and stored in the EntityName policy.

This operation retrieves a **dds::pub::DataWriter** (p. 1138) within a **dds::pub::Publisher** (p. 1993) given the specified name which encodes both to the **dds::pub::DataWriter** (p. 1138) and the **dds::pub::Publisher** (p. 1993) name.

If there are several **dds::pub::DataWriter** (p. 1138) with the same name within the corresponding **dds::pub::Publisher** (p. 1993) this function returns the first matching occurrence.

The specified name might be given as a fully-qualified entity name or as a plain name.

The fully qualified entity name is a concatenation of the **dds::pub::Publisher** (p. 1993) to which the **dds::pub::Data↩ Writer** (p. 1138) belongs and the entity name of of the **dds::pub::DataWriter** (p. 1138) itself, separated by a double colon "::". For example: MyPublisherName::MyDataWriterName

The plain name contains the **dds::pub::DataWriter** (p. 1138) name only. In this situation it is implied that the **dds::pub↩ ::DataWriter** (p. 1138) belongs to the implicit **dds::pub::Publisher** (p. 1993) so the use of a plain name is equivalent to specifying a fully qualified name with the **dds::pub::Publisher** (p. 1993) name part being "implicit". For example: the plain name "MyDataWriterName" is equivalent to specifiying the fully qualified name "implicit::MyDataWriterName"

The **dds::pub::DataWriter** (p. 1138) is only looked up within the **dds::pub::Publisher** (p. 1993) specified in the fully qualified name, or within the implicit **dds::pub::Publisher** (p. 1993) if the name was not fully qualified.

Template Parameters

Writer	The type of the writer. It can be **dds::pub::AnyDataWriter** (p. 815), or an instantiation of dds::pub::DataWriter<T> (if T is not the correct type, this function throws **dds::core::InvalidDowncastError** (p. 1620))

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) within which the **dds::pub::DataWriter** (p. 1138) exists
datawriter_name	**EntityName** (p. 1530) of the **DataWriter** (p. 1138) to find

Returns

The WRITER with the given name

See also

implicit_publisher(const dds::domain::DomainParticipant& dp) (p. 2005);

Looking up DataWriters (p. 314)

References **dds::core::null**.

8.68 rti::core::status::DataWriterCacheStatus Class Reference

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::datawriter_cache()** (p. 2386)

```
#include <Status.hpp>
```

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- int64_t **sample_count** () const

  The number of samples in the DataWriter's queue. This statistic includes meta-samples that represent the unregistration or disposal of an instance.
- int64_t **sample_count_peak** () const

  *The highest value of **rti::core::status::DataWriterCacheStatus::sample_count** (p. 1200) over the lifetime of the DataWriter.*
- int64_t **alive_instance_count** () const

  *The number of instances currently in the DataWriter's queue that have an instance_state equal to **dds::sub::status::InstanceState::alive()** (p. 1617).*
- int64_t **alive_instance_count_peak** () const

  *The highest value of **rti::core::status::DataWriterCacheStatus::alive_instance_count** (p. 1200) over the lifetime of the DataWriter.*

- int64_t **disposed_instance_count** () const

  *The number of instances currently in the DataWriter's queue that have an instance_state equal to **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) (due to, for example, being disposed via the **dds::pub::DataWriter::dispose_instance()** (p. 1162) operation).*
- int64_t **disposed_instance_count_peak** () const

  *The highest value of **rti::core::status::DataWriterCacheStatus::disposed_instance_count** (p. 1201) over the lifetime of the DataWriter.*
- int64_t **unregistered_instance_count** () const

  *The number of instances currently in the DataWriter's queue that the DataWriter has unregistered from via the **dds::pub::DataWriter::unregister_instance** (p. 1160) operation.*
- int64_t **unregistered_instance_count_peak** () const

  *The highest value of **rti::core::status::DataWriterCacheStatus::unregistered_instance_count** (p. 1201) over the lifetime of the DataWriter.*

8.68.1 Detailed Description

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::datawriter_cache()** (p. 2386)

Entity:

dds::pub::DataWriter (p. 1138)

8.68.2 Member Function Documentation

8.68.2.1 sample_count()

```
int64_t rti::core::status::DataWriterCacheStatus::sample_count ( ) const  [inline]
```

The number of samples in the DataWriter's queue. This statistic includes meta-samples that represent the unregistration or disposal of an instance.

8.68.2.2 sample_count_peak()

```
int64_t rti::core::status::DataWriterCacheStatus::sample_count_peak ( ) const  [inline]
```

The highest value of **rti::core::status::DataWriterCacheStatus::sample_count** (p. 1200) over the lifetime of the DataWriter.

8.68.2.3 alive_instance_count()

```
int64_t rti::core::status::DataWriterCacheStatus::alive_instance_count ( ) const  [inline]
```

The number of instances currently in the DataWriter's queue that have an instance_state equal to **dds::sub::status::InstanceState::alive()** (p. 1617).

8.68.2.4 alive_instance_count_peak()

```
int64_t rti::core::status::DataWriterCacheStatus::alive_instance_count_peak ( ) const  [inline]
```

The highest value of **rti::core::status::DataWriterCacheStatus::alive_instance_count** (p. 1200) over the lifetime of the DataWriter.

8.68.2.5 disposed_instance_count()

```
int64_t rti::core::status::DataWriterCacheStatus::disposed_instance_count ( ) const  [inline]
```

The number of instances currently in the DataWriter's queue that have an instance_state equal to **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) (due to, for example, being disposed via the **dds::pub::DataWriter::dispose_instance()** (p. 1162) operation).

8.68.2.6 disposed_instance_count_peak()

```
int64_t rti::core::status::DataWriterCacheStatus::disposed_instance_count_peak ( ) const  [inline]
```

The highest value of **rti::core::status::DataWriterCacheStatus::disposed_instance_count** (p. 1201) over the lifetime of the DataWriter.

8.68.2.7 unregistered_instance_count()

```
int64_t rti::core::status::DataWriterCacheStatus::unregistered_instance_count ( ) const  [inline]
```

The number of instances currently in the DataWriter's queue that the DataWriter has unregistered from via the **dds::pub::DataWriter::unregister_instance** (p. 1160) operation.

8.68.2.8 unregistered_instance_count_peak()

```
int64_t rti::core::status::DataWriterCacheStatus::unregistered_instance_count_peak ( ) const  [inline]
```

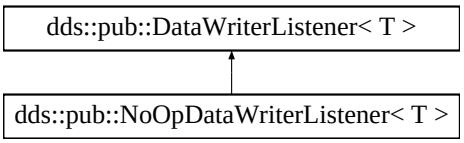
The highest value of **rti::core::status::DataWriterCacheStatus::unregistered_instance_count** (p. 1201) over the lifetime of the DataWriter.

8.69 dds::pub::DataWriterListener< T > Class Template Reference

The **Listener** (p. 1637) to notify status changes for a **dds::pub::DataWriter** (p. 1138).

```
#include <dds/pub/DataWriterListener.hpp>
```

Inheritance diagram for dds::pub::DataWriterListener< T >:



Public Member Functions

- virtual void **on_offered_deadline_missed** (**dds::pub::DataWriter**< T > &writer, const **dds::core::status::OfferedDeadlineMissedStatus** &status)=0

  *Handles the **dds::core::status::OfferedDeadlineMissedStatus** (p. 1871) status.*
- virtual void **on_offered_incompatible_qos** (**dds::pub::DataWriter**< T > &writer, const **dds::core::status::OfferedIncompatibleQosStatus** &status)=0

  *Handles the **dds::core::status::OfferedIncompatibleQosStatus** (p. 1873) status.*
- virtual void **on_liveliness_lost** (**dds::pub::DataWriter**< T > &writer, const **dds::core::status::LivelinessLostStatus** &status)=0

  *Handles the **dds::core::status::LivelinessLostStatus** (p. 1655) status.*
- virtual void **on_publication_matched** (**dds::pub::DataWriter**< T > &writer, const **dds::core::status::PublicationMatchedStatus** &status)=0

  *Handles the **dds::core::status::PublicationMatchedStatus** (p. 1990) status.*
- virtual void **on_reliable_writer_cache_changed** (**dds::pub::DataWriter**< T > &writer, const **rti::core::status::ReliableWriterCacheChangedStatus** &status)=0

  *<<**extension**>> (p. 354) Handles the dds::core::status::ReliableWriterCacheChangedStatus status*
- virtual void **on_reliable_reader_activity_changed** (**dds::pub::DataWriter**< T > &writer, const **rti::core::status::ReliableReaderActivityChangedStatus** &status)=0

  *<<**extension**>> (p. 354) Handles the dds::core::status::ReliableReaderActivityChangedStatus status*
- virtual void **on_instance_replaced** (**dds::pub::DataWriter**< T > &writer, const **dds::core::InstanceHandle** &handle)=0

  *<<**extension**>> (p. 354) Notifies when an instance is replaced in **DataWriter** (p. 1138) queue.*
- virtual void **on_application_acknowledgment** (**dds::pub::DataWriter**< T > &writer, const **rti::pub::AcknowledgmentInfo** &acknowledgment_info)=0

<<***extension***>> (p. *354) Called when a sample is application-acknowledged.*

- virtual void **on_service_request_accepted** (**dds::pub::DataWriter**< T > &writer, const **rti::core::status::ServiceRequestAcceptedStatus** &status)=0

  <<***extension***>> (p. *354) Called when a sample that has been received on a the built-in service channel is intended for the **DataWriter** (p. 1138) that has installed this listener*

- virtual void **on_sample_removed** (**dds::pub::DataWriter**< T > &writer, const **rti::core::Cookie** &cookie)=0

  <<***extension***>> (p. *354) Called when a sample is removed from the **DataWriter** (p. 1138) queue.*

8.69.1 Detailed Description

template<typename T>
class dds::pub::DataWriterListener< T >

The **Listener** (p. 1637) to notify status changes for a **dds::pub::DataWriter** (p. 1138).

Entity:

dds::pub::DataWriter (p. 1138)

Status:

dds::core::status::StatusMask::liveliness_lost() (p. 2383), **dds::core::status::LivelinessLostStatus** (p. 1655);
dds::core::status::StatusMask::offered_deadline_missed() (p. 2379), **dds::core::status::OfferedDeadlineMissedStatus** (p. 1871);
dds::core::status::StatusMask::offered_incompatible_qos() (p. 2380), **dds::core::status::OfferedIncompatibleQosStatus** (p. 1873);
dds::core::status::StatusMask::publication_matched() (p. 2384), **dds::core::status::PublicationMatchedStatus** (p. 1990);
dds::core::status::StatusMask::reliable_reader_activity_changed() (p. 2385), **rti::core::status::ReliableReaderActivityChangedStatus** (p. 2157);
dds::core::status::StatusMask::reliable_writer_cache_changed() (p. 2385), **rti::core::status::ReliableWriterCacheChangedStatus** (p. 2159);

See also

Listener (p. 1637)

Status Kinds (p. 429)

Operations Allowed in Listener Callbacks (p. 1639)

NoOpDataWriterListener (p. 1841)

8.69.2 Member Function Documentation

8.69.2.1 on_offered_deadline_missed()

```
template<typename T >
virtual void  dds::pub::DataWriterListener< T >::on_offered_deadline_missed (
              dds::pub::DataWriter< T > & writer,
             const  dds::core::status::OfferedDeadlineMissedStatus & status )  [pure virtual]
```

Handles the **dds::core::status::OfferedDeadlineMissedStatus** (p. 1871) status.

This callback is called when the deadline that the **dds::pub::DataWriter** (p. 1138) has committed through its **DEADLINE** (p. 523) qos policy was not respected for a specific instance. This callback is called for each deadline period elapsed during which the **dds::pub::DataWriter** (p. 1138) failed to provide data for an instance.

Parameters

writer	<<***out***>> *(p. 355)* Locally created **dds::pub::DataWriter** (p. 1138) that triggers the listener callback
status	<<***out***>> *(p. 355)* Current deadline missed status of locally created **dds::pub::DataWriter** (p. 1138)

Implemented in **dds::pub::NoOpDataWriterListener**< **T** > (p. 1842).

8.69.2.2 on_offered_incompatible_qos()

```
template<typename T >
virtual void  dds::pub::DataWriterListener< T >::on_offered_incompatible_qos (
              dds::pub::DataWriter< T > & writer,
             const  dds::core::status::OfferedIncompatibleQosStatus & status )  [pure virtual]
```

Handles the **dds::core::status::OfferedIncompatibleQosStatus** (p. 1873) status.

This callback is called when the **dds::pub::qos::DataWriterQos** (p. 1226) of the **dds::pub::DataWriter** (p. 1138) was incompatible with what was requested by a **dds::sub::DataReader** (p. 976). This callback is called when a **dds::pub::DataWriter** (p. 1138) has discovered a **dds::sub::DataReader** (p. 976) for the same **dds::topic::Topic** (p. 2474) and common partition, but with a requested QoS that is incompatible with that offered by the **dds::pub::DataWriter** (p. 1138).

Parameters

writer	<<***out***>> *(p. 355)* Locally created **dds::pub::DataWriter** (p. 1138) that triggers the listener callback
status	<<***out***>> *(p. 355)* Current incompatible qos status of locally created **dds::pub::DataWriter** (p. 1138)

Implemented in **dds::pub::NoOpDataWriterListener**< **T** > (p. 1842).

8.69.2.3 on_liveliness_lost()

```
template<typename T >
virtual void  dds::pub::DataWriterListener< T >::on_liveliness_lost (
              dds::pub::DataWriter< T > & writer,
             const  dds::core::status::LivelinessLostStatus & status )  [pure virtual]
```

Handles the **dds::core::status::LivelinessLostStatus** (p. 1655) status.

This callback is called when the liveliness that the **dds::pub::DataWriter** (p. 1138) has committed through its **LIVELINESS** (p. 535) qos policy was not respected; this **dds::sub::DataReader** (p. 976) entities will consider the **dds::pub::DataWriter** (p. 1138) as no longer "alive/active". This callback will not be called when an already not alive **dds::pub::DataWriter** (p. 1138) simply remains not alive for another liveliness period.

Parameters

writer	<<***out***>> *(p. 355)* Locally created **dds::pub::DataWriter** (p. 1138) that triggers the listener callback
status	<<***out***>> *(p. 355)* Current liveliness lost status of locally created **dds::pub::DataWriter** (p. 1138)

Implemented in **dds::pub::NoOpDataWriterListener< T >** (p. 1842).

8.69.2.4 on_publication_matched()

```
template<typename T >
virtual void  dds::pub::DataWriterListener< T >::on_publication_matched (
              dds::pub::DataWriter< T > & writer,
             const  dds::core::status::PublicationMatchedStatus & status )  [pure virtual]
```

Handles the **dds::core::status::PublicationMatchedStatus** (p. 1990) status.

This callback is called when the **dds::pub::DataWriter** (p. 1138) has found a **dds::sub::DataReader** (p. 976) that matches the **dds::topic::Topic** (p. 2474), has a common partition and compatible QoS, or has ceased to be matched with a **dds::sub::DataReader** (p. 976) that was previously considered to be matched.

Parameters

writer	<<***out***>> *(p. 355)* Locally created **dds::pub::DataWriter** (p. 1138) that triggers the listener callback
status	<<***out***>> *(p. 355)* Current publication match status of locally created **dds::pub::DataWriter** (p. 1138)

Implemented in **dds::pub::NoOpDataWriterListener< T >** (p. 1842).

8.69.2.5 on_reliable_writer_cache_changed()

```
template<typename T >
virtual void  dds::pub::DataWriterListener< T >::on_reliable_writer_cache_changed (
```

```
dds::pub::DataWriter< T > & writer,
const rti::core::status::ReliableWriterCacheChangedStatus & status ) [pure virtual]
```

<<***extension***>> *(p. 354)* Handles the dds::core::status::ReliableWriterCacheChangedStatus status

This listener callback is triggered when:

- The cache is empty (contains no unacknowledged samples).
- The cache is full (the number of unacknowledged samples has reached the value specified in **dds::core::policy::ResourceLimits::max_samples** (p. 2205)).
- The number of unacknowledged samples has reached **rti::core::policy::RtpsReliableWriterProtocol::high_watermark** (p. 2233) or **rti::core::policy::RtpsReliableWriterProtocol::low_watermark** (p. 2232).

Parameters

writer	<<***out***>> *(p. 355)* Locally created **dds::pub::DataWriter** (p. 1138) that triggers the listener callback
status	<<***out***>> *(p. 355)* Current reliable writer cache changed status of locally created **dds::pub::DataWriter** (p. 1138)

Implemented in **dds::pub::NoOpDataWriterListener< T >** (p. 1843).

8.69.2.6 on_reliable_reader_activity_changed()

```
template<typename T >
virtual void dds::pub::DataWriterListener< T >::on_reliable_reader_activity_changed (
dds::pub::DataWriter< T > & writer,
const rti::core::status::ReliableReaderActivityChangedStatus & status ) [pure virtual]
```

<<***extension***>> *(p. 354)* Handles the dds::core::status::ReliableReaderActivityChangedStatus status

Parameters

writer	<<***out***>> *(p. 355)* Locally created **dds::pub::DataWriter** (p. 1138) that triggers the listener callback
status	<<***out***>> *(p. 355)* Current reliable reader activity changed status of locally created **dds::pub::DataWriter** (p. 1138)

Implemented in **dds::pub::NoOpDataWriterListener< T >** (p. 1843).

8.69.2.7 on_instance_replaced()

```
template<typename T >
virtual void dds::pub::DataWriterListener< T >::on_instance_replaced (
```

```
        dds::pub::DataWriter< T > & writer,
    const  dds::core::InstanceHandle & handle )  [pure virtual]
```

<<***extension***>> *(p. 354)* Notifies when an instance is replaced in **DataWriter** (p. 1138) queue.

This callback is called when an instance is replaced by the **dds::pub::DataWriter** (p. 1138) due to instance resource limits being reached. This callback returns to the user the handle of the replaced instance, which can be used to get the key of the replaced instance using the **dds::pub::DataWriter::key_value** (p. 1164) API.

Because this callback can be called within the context of an in-progress write, dispose, or unregister call, most APIs on the **DataWriter** (p. 1138) must not be used. The only **DataWriter** (p. 1138) APIs that are safe to call within this callback are:

- **dds::pub::DataWriter::key_value** (p. 1164)
- **dds::pub::DataWriter::create_data** (p. 1182)
- **dds::pub::DataWriter::delete_data** (p. 1183)
- **dds::pub::matched_subscriptions()** (p. 648)
- **dds::pub::DataWriter::is_matched_subscription_active** (p. 1191)
- **dds::pub::DataWriter::matched_subscription_participant_data()** (p. 1190)
- **dds::pub::DataWriter::topic()** (p. 1168)
- **dds::pub::DataWriter::publisher()** (p. 1168)
- **dds::pub::DataWriter::is_sample_app_acknowledged** (p. 1173)

Parameters

writer	<<***out***>> *(p. 355)* Locally created **dds::pub::DataWriter** (p. 1138) that triggers the listener callback
handle	<<***out***>> *(p. 355)* Handle of the replaced instance

Implemented in **dds::pub::NoOpDataWriterListener**< **T** > (p. 1843).

8.69.2.8 on_application_acknowledgment()

```
template<typename T >
virtual void  dds::pub::DataWriterListener< T >::on_application_acknowledgment (
        dds::pub::DataWriter< T > & writer,
    const  rti::pub::AcknowledgmentInfo & acknowledgment_info )  [pure virtual]
```

<<***extension***>> *(p. 354)* Called when a sample is application-acknowledged.

Applicable only when **dds::core::policy::Reliability::acknowledgment_kind** (p. 2154) = **rti::core::policy::AcknowledgmentKind_def::APPLICATION_AUTO** (p. 798) or **rti::core::policy::AcknowledgmentKind_def::APPLICATION_EXPLICIT** (p. 798)

Called when a sample is application-level acknowledged. Provides identities of the sample and the acknowledging **dds::sub::DataReader** (p. 976). Also provides user-specified response data sent from the **dds::sub::DataReader** (p. 976) by the acknowledgment message.

Parameters

writer	<<***out***>> *(*p. *355)* Locally created **DataWriter** (p. 1138) that triggers the listener callback
acknowledgment_info	<<***out***>> *(*p. *355)* **rti::pub::AcknowledgmentInfo** (p. 795) of the acknowledged sample

Implemented in **dds::pub::NoOpDataWriterListener< T >** (p. 1843).

8.69.2.9 on_service_request_accepted()

```
template<typename T >
virtual void  dds::pub::DataWriterListener< T >::on_service_request_accepted (
              dds::pub::DataWriter< T > & writer,
             const  rti::core::status::ServiceRequestAcceptedStatus & status )  [pure virtual]
```

<<***extension***>> *(*p. *354)* Called when a sample that has been received on a the built-in service channel is intended for the **DataWriter** (p. 1138) that has installed this listener

See also

Topic Queries (p. 67)

Implemented in **dds::pub::NoOpDataWriterListener< T >** (p. 1844).

8.69.2.10 on_sample_removed()

```
template<typename T >
virtual void  dds::pub::DataWriterListener< T >::on_sample_removed (
              dds::pub::DataWriter< T > & writer,
             const  rti::core::Cookie & cookie )  [pure virtual]
```

<<***extension***>> *(*p. *354)* Called when a sample is removed from the **DataWriter** (p. 1138) queue.

This callback is called only if the sample was written with a **rti::core::Cookie** (p. 965) with **dds::pub::DataWriter↩ ::write(const T&,rti::pub::WriteParams&)** (p. 1179), or if this writer uses **Zero Copy** (p. 50) transfer over shared memory" or **FlatData Topic-Types** (p. 419) "FlatData language binding".

Parameters

writer	<<***out***>> *(*p. *355)* Locally created **dds::pub::DataWriter** (p. 1138) that triggers the listener callback
cookie	<<***out***>> *(*p. *355)* • If this sample was written with **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179), this field contains a copy of the cookie set in **rti::pub::WriteParams** (p. 2642). • If this writer uses **Zero Copy transfer over shared memory** (p. 50) or **FlatData language binding** (p. 419), this field contains the absolute address of the sample that is removed. The address of the sample can be obtained by using **rti::core::Cookie::to_pointer** (p. 966)

See also

dds::pub::DataWriter::get_loan (p. 1183)

Implemented in **dds::pub::NoOpDataWriterListener**< **T** > (p. 1845).

8.70 rti::core::policy::DataWriterProtocol Class Reference

<<***extension***>> *(p. 116)* Configures aspects of an the RTPS protocol related to a **dds::pub::DataWriter** (p. 1138)

`#include <rti/core/policy/CorePolicy.hpp>`

Public Member Functions

- **DataWriterProtocol** ()
  Creates the default policy.
- **DataWriterProtocol** & **virtual_guid** (const **rti::core::Guid** &the_virtual_guid)
  <<***extension***>> *(p. 116) The virtual GUID (Global Unique Identifier).*
- **rti::core::Guid virtual_guid** () const
  Getter (see setter with the same name)
- **DataWriterProtocol** & **rtps_object_id** (uint32_t the_rtps_object_id)
  <<***extension***>> *(p. 116) The RTPS Object ID.*
- uint32_t **rtps_object_id** () const
  Getter (see setter with the same name)
- **DataWriterProtocol** & **push_on_write** (bool the_push_on_write)
  <<***extension***>> *(p. 116) Whether to push sample out when write is called.*
- bool **push_on_write** () const
  Getter (see setter with the same name)
- **DataWriterProtocol** & **disable_positive_acks** (bool the_disable_positive_acks)
  <<***extension***>> *(p. 116) Controls whether or not the writer expects positive acknowledgements from matching readers.*
- bool **disable_positive_acks** () const
  Getter (see setter with the same name)
- **DataWriterProtocol** & **disable_inline_keyhash** (bool the_disable_inline_keyhash)
  <<***extension***>> *(p. 116) Controls whether or not a keyhash is propagated on the wire with each sample.*
- bool **disable_inline_keyhash** () const
  Getter (see setter with the same name)
- **DataWriterProtocol** & **serialize_key_with_dispose** (bool the_serialize_key_with_dispose)
  <<***extension***>> *(p. 116) Controls whether or not the serialized key is propagated on the wire with dispose samples.*
- bool **serialize_key_with_dispose** () const
  Getter (see setter with the same name)
- **DataWriterProtocol** & **propagate_app_ack_with_no_response** (bool the_propagate_app_ack_with_no_response)
  <<***extension***>> *(p. 116) Controls whether or not a* ***dds::pub::DataWriter*** *(p. 1138) receives* ***dds::pub::DataWriterListener::on_application_acknowledgment*** *(p. 1207) notifications with an empty or invalid response.*
- bool **propagate_app_ack_with_no_response** () const
  Getter (see setter with the same name)

- **DataWriterProtocol** & **rtps_reliable_writer** (const **RtpsReliableWriterProtocol** &the_rtps_reliable_writer)

  <<***extension***>> *(p. 116) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a* ***dds::pub::DataWriter*** *(p. 1138). This parameter only has effect if both the writer and the matching reader are configured with* ***dds::core::policy::ReliabilityKind_def::RELIABLE*** *(p. 2157)* ***dds::core::policy::ReliabilityKind_def*** *(p. 2156).*
- const **RtpsReliableWriterProtocol** & **rtps_reliable_writer** () const

  Gets the reliable settings by const-reference (see setter)
- **RtpsReliableWriterProtocol** & **rtps_reliable_writer** ()

  Gets the reliable settings by reference (see setter)
- **DataWriterProtocol** & **initial_virtual_sequence_number** (const **rti::core::SequenceNumber** &the_initial_↩ virtual_sequence_number)

  <<***extension***>> *(p. 116) Determines, the initial virtual sequence number for this DataWriter.*

8.70.1 Detailed Description

<<***extension***>> *(p. 116)* Configures aspects of an the RTPS protocol related to a **dds::pub::DataWriter** (p. 1138)

DDS has a standard protocol for packet (user and meta data) exchange between applications using DDS for communications. This QoS policy and **rti::core::policy::DataWriterProtocol** (p. 1210) give you control over configurable portions of the protocol, including the configuration of the reliable data delivery mechanism of the protocol on a per DataWriter or DataReader basis.

These configuration parameters control timing, timeouts, and give you the ability to tradeoff between speed of data loss detection and repair versus network and CPU bandwidth used to maintain reliability.

It is important to tune the reliability protocol (on a per **dds::pub::DataWriter** (p. 1138) and **dds::sub::DataReader** (p. 976) basis) to meet the requirements of the end-user application so that data can be sent between DataWriters and DataReaders in an efficient and optimal manner in the presence of data loss.

You can also use this QoS policy to control how RTI Connext responds to "slow" reliable DataReaders or ones that disconnect or are otherwise lost. See **dds::core::policy::Reliability** (p. 2149) for more information on the per-DataReader/DataWriter reliability configuration. **dds::core::policy::History** (p. 1602) and **dds::core::policy::ResourceLimits** (p. 2203) also play an important role in the DDS reliable protocol.

This QoS policy is an extension to the DDS standard.

Entity:

dds::pub::DataWriter (p. 1138)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.70.2 Constructor & Destructor Documentation

8.70.2.1 DataWriterProtocol()

```
rti::core::policy::DataWriterProtocol::DataWriterProtocol ( )  [inline]
```

Creates the default policy.

8.70.3 Member Function Documentation

8.70.3.1 virtual_guid() [1/2]

```
DataWriterProtocol & rti::core::policy::DataWriterProtocol::virtual_guid (
            const  rti::core::Guid & the_virtual_guid )
```

<<***extension***>> *(*p. *116)* The virtual GUID (Global Unique Identifier).

The virtual GUID is used to uniquely identify different incarnations of the same **dds::pub::DataWriter** (p. 1138).

RTI Connext uses the virtual GUID to associate a persisted writer history to a specific **dds::pub::DataWriter** (p. 1138).

The RTI Connext Persistence **Service** (p. 2350) uses the virtual GUID to send samples on behalf of the original **dds::pub::DataWriter** (p. 1138).

[default] rti::core::Guid::automatic() (p. 1597)

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Data Availability** (p. 167)

8.70.3.2 virtual_guid() [2/2]

```
rti::core::Guid rti::core::policy::DataWriterProtocol::virtual_guid ( ) const
```

Getter (see setter with the same name)

8.70.3.3 rtps_object_id() [1/2]

```
DataWriterProtocol & rti::core::policy::DataWriterProtocol::rtps_object_id (
            uint32_t the_rtps_object_id )
```

<<***extension***>> *(p. 116)* The RTPS Object ID.

This value is used to determine the RTPS object ID of a data writer according to the DDS-RTPS Interoperability Wire Protocol.

Only the last 3 bytes are used; the most significant byte is ignored.

If the default value is specified, RTI Connext will automatically assign the object ID based on a counter value (per participant) starting at 0x00800000. That value is incremented for each new data writer.

A rtps_object_id value in the interval [0x00800000,0x00ffffff] may collide with the automatic values assigned by RTI Connext. In those cases, the recomendation is not to use automatic object ID assigment.

[default] rti::core::policy::WireProtocol::RTPS_AUTO_ID (p. 2641)

[range] [0,0x00ffffff]

Categories

Immutable (p. 243)

8.70.3.4 rtps_object_id() [2/2]

```
uint32_t rti::core::policy::DataWriterProtocol::rtps_object_id ( ) const
```

Getter (see setter with the same name)

8.70.3.5 push_on_write() [1/2]

```
DataWriterProtocol & rti::core::policy::DataWriterProtocol::push_on_write (
            bool the_push_on_write )
```

<<***extension***>> *(p. 116)* Whether to push sample out when write is called.

If set to true (the default), the writer will send a sample every time write is called. Otherwise, the sample is put into the queue waiting for a NACK from remote reader(s) to be sent out.

[default] true

Categories

Immutable (p. 243), **Reliability** (p. 222), **Data Availability** (p. 167)

8.70.3.6 push_on_write() [2/2]

```
bool rti::core::policy::DataWriterProtocol::push_on_write ( ) const
```

Getter (see setter with the same name)

8.70.3.7 disable_positive_acks() [1/2]

```
DataWriterProtocol & rti::core::policy::DataWriterProtocol::disable_positive_acks (
            bool the_disable_positive_acks )
```

<<***extension***>> *(*p. *116)* Controls whether or not the writer expects positive acknowledgements from matching readers.

If set to true, the writer does not expect readers to send send positive acknowledgments to the writer. Consequently, instead of keeping a sample queued until all readers have positively acknowledged it, the writer will keep a sample for at least **rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_min_sample_keep_duration** (p. 2245), after which the sample is logically considered as positively acknowledged.

If set to false (the default), the writer expects to receive positive acknowledgements from its acknowledging readers (**rti::core::policy::DataReaderProtocol::disable_positive_acks** (p. 1062) = false) and it applies the keep-duration to its non-acknowledging readers (**rti::core::policy::DataReaderProtocol::disable_positive_acks** (p. 1062) = true).

A writer with both acknowledging and non-acknowledging readers keeps a sample queued until acknowledgements have been received from all acknowledging readers and the keep-duration has elapsed for non-acknowledging readers.

[default] false

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Reliability** (p. 222)

8.70.3.8 disable_positive_acks() [2/2]

```
bool rti::core::policy::DataWriterProtocol::disable_positive_acks ( ) const
```

Getter (see setter with the same name)

8.70.3.9 disable_inline_keyhash() [1/2]

```
DataWriterProtocol & rti::core::policy::DataWriterProtocol::disable_inline_keyhash (
            bool the_disable_inline_keyhash )
```

<<***extension***>> *(*p. *116)* Controls whether or not a keyhash is propagated on the wire with each sample.

This field only applies to keyed writers.

With each key, RTI Connext associates an internal 16-byte representation, called a keyhash.

When this field is false, the keyhash is sent on the wire with every data instance.

When this field is true, the keyhash is not sent on the wire and the readers must compute the value using the received data.

If the *reader* is CPU bound, sending the keyhash on the wire may increase performance, because the reader does not have to get the keyhash from the data.

If the *writer* is CPU bound, sending the keyhash on the wire may decrease performance, because it requires more bandwidth (16 more bytes per sample).

Note: Setting disable_inline_keyhash to true is not compatible with using RTI Real-Time Connect or RTI Recorder.

[default] false

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193)

8.70.3.10 disable_inline_keyhash() [2/2]

```
bool rti::core::policy::DataWriterProtocol::disable_inline_keyhash ( ) const
```

Getter (see setter with the same name)

8.70.3.11 serialize_key_with_dispose() [1/2]

```
DataWriterProtocol & rti::core::policy::DataWriterProtocol::serialize_key_with_dispose (
            bool the_serialize_key_with_dispose )
```

<<***extension***>> *(*p. *116)* Controls whether or not the serialized key is propagated on the wire with dispose samples.

This field only applies to keyed writers.

We recommend setting this field to true if there are DataReaders where **rti::core::policy::DataReaderProtocol**↩ **::propagate_dispose_of_unregistered_instances** (p. 1063) is also true.

When setting serialize_key_with_dispose to FALSE, only a key hash is included in the dispose meta-sample sent by a DataWriter for a dispose action. If a dispose meta-sample only includes the key hash, then DataReaders must have previously received an actual data sample for the instance being disposed, in order for a DataReader to map a key hash/instance handle to actual key values.

If an actual data sample was never received for an instance and serialize_key_with_dispose is set to FALSE, then the DataReader application will not be able to determine the value of the key that was disposed, since **dds::sub::Data**↩ **Reader::key_value** (p. 1003) will not be able to map an instance handle to actual key values.

By setting serialize_key_with_dispose to TRUE, the values of the key members of a data type will be sent in the dispose meta-sample for a dispose action by the DataWriter. This allows the DataReader to map an instance handle to the values of the key members even when receiving a dispose meta-sample without previously having received a data sample for the instance.

Important: When this field is true, batching will not be compatible with RTI Connext 4.3e, 4.4b, or 4.4c. The **dds::sub**↩ **::DataReader** (p. 976) entities will receive incorrect data and/or encounter deserialization errors.

[default] false

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193)

8.70.3.12 serialize_key_with_dispose() [2/2]

```
bool rti::core::policy::DataWriterProtocol::serialize_key_with_dispose ( ) const
```

Getter (see setter with the same name)

8.70.3.13 propagate_app_ack_with_no_response() [1/2]

```
DataWriterProtocol & rti::core::policy::DataWriterProtocol::propagate_app_ack_with_no_response (
            bool the_propagate_app_ack_with_no_response )
```

<<***extension***>> *(p. 116)* Controls whether or not a **dds::pub::DataWriter** (p. 1138) receives **dds::pub::DataWriter↩ Listener::on_application_acknowledgment** (p. 1207) notifications with an empty or invalid response.

When this field is set to false, the callback **dds::pub::DataWriterListener::on_application_acknowledgment** (p. 1207) will not be invoked if the sample being acknowledged has an empty or invalid response.

[default] true

Categories

Immutable (p. 243), **Reliability** (p. 222), **Application Acknowledgement** (p. 158), **Durability** (p. 184)

8.70.3.14 propagate_app_ack_with_no_response() [2/2]

```
bool rti::core::policy::DataWriterProtocol::propagate_app_ack_with_no_response ( ) const
```

Getter (see setter with the same name)

8.70.3.15 rtps_reliable_writer() [1/3]

```
DataWriterProtocol & rti::core::policy::DataWriterProtocol::rtps_reliable_writer (
            const  RtpsReliableWriterProtocol & the_rtps_reliable_writer )
```

<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **dds::pub::DataWriter** (p. 1138). This parameter only has effect if both the writer and the matching reader are configured with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core::policy::ReliabilityKind_def** (p. 2156).

For details, refer to the **rti::core::policy::RtpsReliableWriterProtocol** (p. 2229)

[default] [default] See **rti::core::policy::RtpsReliableWriterProtocol** (p. 2229)

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.70.3.16 rtps_reliable_writer() [2/3]

```
const RtpsReliableWriterProtocol & rti::core::policy::DataWriterProtocol::rtps_reliable_writer (
) const
```

Gets the reliable settings by const-reference (see setter)

8.70.3.17 rtps_reliable_writer() [3/3]

```
RtpsReliableWriterProtocol & rti::core::policy::DataWriterProtocol::rtps_reliable_writer ( )
```

Gets the reliable settings by reference (see setter)

8.70.3.18 initial_virtual_sequence_number()

```
DataWriterProtocol & rti::core::policy::DataWriterProtocol::initial_virtual_sequence_number (
            const rti::core::SequenceNumber & the_initial_virtual_sequence_number )
```

<<***extension***>> *(*p. *116)* Determines, the initial virtual sequence number for this DataWriter.

By default, the virtual sequence number of the first sample published by a DataWriter will be 1 for DataWriters that do not use durable writer history. For durable writers, the default virtual sequence number will be the last sequence number they published in a previous execution, plus one. So, when a non-durable DataWriter is restarted and must continue communicating with the same DataReaders, its samples start over with sequence number 1. Durable DataWriters start over where the last sequence number left off, plus one.

This QoS setting allows overwriting the default initial virtual sequence number.

Normally, this parameter is not expected to be modified; however, in some scenarios when continuing communication after restarting, applications may require the DataWriter's virtual sequence number to start at something other than the value described above. An example would be to enable non-durable DataWriters to start at the last sequence number published, plus one, similar to the durable DataWriter. This property enables you to make such a configuration, if desired.

The virtual sequence number can be overwritten as well on a per sample basis by updating **rti::pub::WriteParams↩ ::identity** (p. 2646) in the **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179).

[default] SequenceNumber_t::AUTO

Categories

Immutable (p. 243)

8.71 rti::core::status::DataWriterProtocolStatus Class Reference

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::datawriter_protocol()** (p. 2386)

```
#include <Status.hpp>
```

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **EventCount64 pushed_sample_count** () const

  The number of user samples pushed on write from a local DataWriter to a matching remote DataReader.
- **EventCount64 pushed_sample_bytes** () const

  The number of bytes of user samples pushed on write from a local DataWriter to a matching remote DataReader.
- **EventCount64 filtered_sample_count** () const

  [Not supported.] *The number of user samples preemptively filtered by a local DataWriter due to Content-Filtered Topics.*
- **EventCount64 filtered_sample_bytes** () const

  [Not supported.] *The number of bytes of user samples preemptively filtered by a local DataWriter due to Content-Filtered Topics.*
- **EventCount64 sent_heartbeat_count** () const

  The number of Heartbeats sent between a local DataWriter and matching remote DataReader.
- **EventCount64 sent_heartbeat_bytes** () const

  The number of bytes of Heartbeats sent between a local DataWriter and matching remote DataReader.
- **EventCount64 pulled_sample_count** () const

  The number of user samples pulled from local DataWriter by matching DataReaders.
- **EventCount64 pulled_sample_bytes** () const

  The number of bytes of user samples pulled from local DataWriter by matching DataReaders.
- **EventCount64 received_ack_count** () const

  The number of ACKs from a remote DataReader received by a local DataWriter.
- **EventCount64 received_ack_bytes** () const

  The number of bytes of ACKs from a remote DataReader received by a local DataWriter.
- **EventCount64 received_nack_count** () const

  The number of NACKs from a remote DataReader received by a local DataWriter.
- **EventCount64 received_nack_bytes** () const

  The number of bytes of NACKs from a remote DataReader received by a local DataWriter.
- **EventCount64 sent_gap_count** () const

  The number of GAPs sent from local DataWriter to matching remote DataReaders.
- **EventCount64 sent_gap_bytes** () const

  The number of bytes of GAPs sent from local DataWriter to matching remote DataReaders.
- **EventCount64 rejected_sample_count** () const

  [Not supported.]
- int32_t **send_window_size** () const

  Current maximum number of outstanding samples allowed in the DataWriter's queue.
- **rti::core::SequenceNumber first_available_sample_sequence_number** () const

  The sequence number of the first available sample currently queued in the local DataWriter.
- **rti::core::SequenceNumber last_available_sample_sequence_number** () const

The sequence number of the last available sample currently queued in the local DataWriter.

- **rti::core::SequenceNumber first_unacknowledged_sample_sequence_number** () const

  The sequence number of the first unacknowledged sample currently queued in the local DataWriter.
- **rti::core::SequenceNumber first_available_sample_virtual_sequence_number** () const

  The virtual sequence number of the first available sample currently queued in the local DataWriter.
- **rti::core::SequenceNumber last_available_sample_virtual_sequence_number** () const

  The virtual sequence number of the last available sample currently queued in the local DataWriter.
- **rti::core::SequenceNumber first_unacknowledged_sample_virtual_sequence_number** () const

  The virtual sequence number of the first unacknowledged sample currently queued in the local DataWriter.
- **dds::core::InstanceHandle first_unacknowledged_sample_subscription_handle** () const

  The handle of a remote DataReader that has not acknowledged the first unacknowledged sample of the local DataWriter.
- **rti::core::SequenceNumber first_unelapsed_keep_duration_sample_sequence_number** () const

  The sequence number of the first sample whose keep duration has not yet elapsed.
- int64_t **pushed_fragment_count** () const

  The number of DATA_FRAG messages that have been pushed by this DataWriter.
- int64_t **pushed_fragment_bytes** () const

  The number of bytes of DATA_FRAG messages that have been pushed by this DataWriter.
- int64_t **pulled_fragment_count** () const

  The number of DATA_FRAG messages that have been pulled from this DataWriter.
- int64_t **pulled_fragment_bytes** () const

  The number of bytes of DATA_FRAG messages that have been pulled from this DataWriter.
- int64_t **received_nack_fragment_count** () const

  The number of NACK_FRAG messages that have been received by this DataWriter.
- int64_t **received_nack_fragment_bytes** () const

  The number of bytes of NACK_FRAG messages that have been received by this DataWriter.

8.71.1 Detailed Description

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::datawriter_protocol()** (p. 2386)

Entity:

dds::pub::DataWriter (p. 1138)

8.71.2 Member Function Documentation

8.71.2.1 pushed_sample_count()

EventCount64 `rti::core::status::DataWriterProtocolStatus::pushed_sample_count ( ) const [inline]`

The number of user samples pushed on write from a local DataWriter to a matching remote DataReader.

Counts protocol (RTPS) messages pushed by a DataWriter when writing, unregistering, and disposing. The count is the number of sends done internally, and it may be greater than the number of user writes.

For large data, counts whole samples, not fragments. The fragment count is tracked in the **rti::core::status::Data↩ WriterProtocolStatus::pushed_fragment_count** (p. 1225) statistic.

8.71.2.2 pushed_sample_bytes()

```
EventCount64 rti::core::status::DataWriterProtocolStatus::pushed_sample_bytes ( ) const  [inline]
```

The number of bytes of user samples pushed on write from a local DataWriter to a matching remote DataReader.

Counts bytes of protocol (RTPS) messages pushed by a DataWriter when writing, unregistering, and disposing. The count of bytes corresponds to the number of sends done internally, and it may be greater than the number of user writes.

When data fragmentation is used, this statistic is incremented as fragments are written.

8.71.2.3 filtered_sample_count()

```
EventCount64 rti::core::status::DataWriterProtocolStatus::filtered_sample_count ( ) const  [inline]
```

[Not supported.] The number of user samples preemptively filtered by a local DataWriter due to Content-Filtered Topics.

8.71.2.4 filtered_sample_bytes()

```
EventCount64 rti::core::status::DataWriterProtocolStatus::filtered_sample_bytes ( ) const  [inline]
```

[Not supported.] The number of bytes of user samples preemptively filtered by a local DataWriter due to Content-↩ Filtered Topics.

8.71.2.5 sent_heartbeat_count()

```
EventCount64 rti::core::status::DataWriterProtocolStatus::sent_heartbeat_count ( ) const  [inline]
```

The number of Heartbeats sent between a local DataWriter and matching remote DataReader.

Because periodic and piggyback heartbeats are sent to remote readers and their locators differently in different situations, when a reader has more than one locator, this count may be larger than expected, to reflect the sending of Heartbeats to the multiple locators.

8.71.2.6 sent_heartbeat_bytes()

```
EventCount64 rti::core::status::DataWriterProtocolStatus::sent_heartbeat_bytes ( ) const  [inline]
```

The number of bytes of Heartbeats sent between a local DataWriter and matching remote DataReader.

Because periodic and piggyback heartbeats are sent to remote readers and their locators differently in different situations, when a reader has more than one locator, this count may be larger than expected, to reflect the sending of Heartbeats to the multiple locators.

8.71.2.7 pulled_sample_count()

EventCount64 rti::core::status::DataWriterProtocolStatus::pulled_sample_count ( ) const [inline]

The number of user samples pulled from local DataWriter by matching DataReaders.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local DataWriter when **rti::core::policy::DataWriterProtocol::push_on_write** (p. 1213) is false.

When data fragmentation is used, this statistic is incremented as fragments are written.

8.71.2.8 pulled_sample_bytes()

EventCount64 rti::core::status::DataWriterProtocolStatus::pulled_sample_bytes ( ) const [inline]

The number of bytes of user samples pulled from local DataWriter by matching DataReaders.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local DataWriter when **rti::core::policy::DataWriterProtocol::push_on_write** (p. 1213) is false.

When data fragmentation is used, this statistic is incremented as fragments are written.

8.71.2.9 received_ack_count()

EventCount64 rti::core::status::DataWriterProtocolStatus::received_ack_count ( ) const [inline]

The number of ACKs from a remote DataReader received by a local DataWriter.

8.71.2.10 received_ack_bytes()

EventCount64 rti::core::status::DataWriterProtocolStatus::received_ack_bytes ( ) const [inline]

The number of bytes of ACKs from a remote DataReader received by a local DataWriter.

8.71.2.11 received_nack_count()

EventCount64 rti::core::status::DataWriterProtocolStatus::received_nack_count ( ) const [inline]

The number of NACKs from a remote DataReader received by a local DataWriter.

8.71.2.12 received_nack_bytes()

```
EventCount64 rti::core::status::DataWriterProtocolStatus::received_nack_bytes ( ) const  [inline]
```

The number of bytes of NACKs from a remote DataReader received by a local DataWriter.

8.71.2.13 sent_gap_count()

```
EventCount64 rti::core::status::DataWriterProtocolStatus::sent_gap_count ( ) const  [inline]
```

The number of GAPs sent from local DataWriter to matching remote DataReaders.

8.71.2.14 sent_gap_bytes()

```
EventCount64 rti::core::status::DataWriterProtocolStatus::sent_gap_bytes ( ) const  [inline]
```

The number of bytes of GAPs sent from local DataWriter to matching remote DataReaders.

8.71.2.15 rejected_sample_count()

```
EventCount64 rti::core::status::DataWriterProtocolStatus::rejected_sample_count ( ) const  [inline]
```

[Not supported.]

8.71.2.16 send_window_size()

```
int32_t rti::core::status::DataWriterProtocolStatus::send_window_size ( ) const  [inline]
```

Current maximum number of outstanding samples allowed in the DataWriter's queue.

Spans the range from **rti::core::policy::RtpsReliableWriterProtocol::min_send_window_size** (p. 2249) to **rti↩ ::core::policy::RtpsReliableWriterProtocol::max_send_window_size** (p. 2250).

8.71.2.17 first_available_sample_sequence_number()

```
rti::core::SequenceNumber rti::core::status::DataWriterProtocolStatus::first_available_sample_↩
sequence_number ( ) const  [inline]
```

The sequence number of the first available sample currently queued in the local DataWriter.

Applies only for local DataWriter status.

8.71.2.18 last_available_sample_sequence_number()

```
rti::core::SequenceNumber rti::core::status::DataWriterProtocolStatus::last_available_sample_↩
sequence_number ( ) const  [inline]
```

The sequence number of the last available sample currently queued in the local DataWriter.

Applies only for local DataWriter status.

8.71.2.19 first_unacknowledged_sample_sequence_number()

```
rti::core::SequenceNumber rti::core::status::DataWriterProtocolStatus::first_unacknowledged_↩
sample_sequence_number ( ) const  [inline]
```

The sequence number of the first unacknowledged sample currently queued in the local DataWriter.

Applies only for local DataWriter status.

8.71.2.20 first_available_sample_virtual_sequence_number()

```
rti::core::SequenceNumber rti::core::status::DataWriterProtocolStatus::first_available_sample_↩
virtual_sequence_number ( ) const  [inline]
```

The virtual sequence number of the first available sample currently queued in the local DataWriter.

Applies only for local DataWriter status.

8.71.2.21 last_available_sample_virtual_sequence_number()

```
rti::core::SequenceNumber rti::core::status::DataWriterProtocolStatus::last_available_sample_↩
virtual_sequence_number ( ) const  [inline]
```

The virtual sequence number of the last available sample currently queued in the local DataWriter.

Applies only for local DataWriter status.

8.71.2.22 first_unacknowledged_sample_virtual_sequence_number()

```
rti::core::SequenceNumber rti::core::status::DataWriterProtocolStatus::first_unacknowledged_↩
sample_virtual_sequence_number ( ) const  [inline]
```

The virtual sequence number of the first unacknowledged sample currently queued in the local DataWriter.

Applies only for local DataWriter status.

8.71.2.23 first_unacknowledged_sample_subscription_handle()

```
dds::core::InstanceHandle rti::core::status::DataWriterProtocolStatus::first_unacknowledged_↩
sample_subscription_handle ( ) const  [inline]
```

The handle of a remote DataReader that has not acknowledged the first unacknowledged sample of the local DataWriter.

Applies only for local DataWriter status.

8.71.2.24 first_unelapsed_keep_duration_sample_sequence_number()

```
rti::core::SequenceNumber rti::core::status::DataWriterProtocolStatus::first_unelapsed_keep_↩
duration_sample_sequence_number ( ) const  [inline]
```

The sequence number of the first sample whose keep duration has not yet elapsed.

Applicable only when **rti::core::policy::DataWriterProtocol::disable_positive_acks** (p. 1214) is set.

Sequence number of the first sample kept in the DataWriter's queue whose keep_duration (applied when **rti::core↩ ::policy::DataWriterProtocol::disable_positive_acks** (p. 1214) is set) has not yet elapsed.

Applies only for local DataWriter status.

8.71.2.25 pushed_fragment_count()

```
int64_t rti::core::status::DataWriterProtocolStatus::pushed_fragment_count ( ) const  [inline]
```

The number of DATA_FRAG messages that have been pushed by this DataWriter.

This statistic is incremented as each DATA_FRAG message is sent, not when the entire sample has been sent. Applicable only when data is fragmented.

8.71.2.26 pushed_fragment_bytes()

```
int64_t rti::core::status::DataWriterProtocolStatus::pushed_fragment_bytes ( ) const  [inline]
```

The number of bytes of DATA_FRAG messages that have been pushed by this DataWriter.

This statistic is incremented as each DATA_FRAG message is sent, not when the entire sample has been sent. Applicable only when data is fragmented.

8.71.2.27 pulled_fragment_count()

```
int64_t rti::core::status::DataWriterProtocolStatus::pulled_fragment_count ( ) const  [inline]
```

The number of DATA_FRAG messages that have been pulled from this DataWriter.

This statistic is incremented as each DATA_FRAG message is sent, not when the entire sample has been sent. Applicable only when data is fragmented.

8.71.2.28 pulled_fragment_bytes()

```
int64_t rti::core::status::DataWriterProtocolStatus::pulled_fragment_bytes ( ) const  [inline]
```

The number of bytes of DATA_FRAG messages that have been pulled from this DataWriter.

This statistic is incremented as each DATA_FRAG message is sent, not when the entire sample has been sent. Applicable only when data is fragmented.

8.71.2.29 received_nack_fragment_count()

```
int64_t rti::core::status::DataWriterProtocolStatus::received_nack_fragment_count ( ) const  [inline]
```

The number of NACK_FRAG messages that have been received by this DataWriter.

NACK FRAG RTPS messages are sent when large data is used in conjunction with reliable communication. They have the same properties as NACK messages, but instead of applying to samples, they apply to fragments. Applicable only when data is fragmented.

8.71.2.30 received_nack_fragment_bytes()

```
int64_t rti::core::status::DataWriterProtocolStatus::received_nack_fragment_bytes ( ) const  [inline]
```

The number of bytes of NACK_FRAG messages that have been received by this DataWriter.

NACK FRAG RTPS messages are sent when large data is used in conjunction with reliable communication. They have the same properties as NACK messages, but instead of applying to samples, they apply to fragments. Applicable only when data is fragmented.

8.72 dds::pub::qos::DataWriterQos Class Reference

<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::pub::DataWriter** (p. 1138) supports

```
#include <dds/pub/qos/DataWriterQos.hpp>
```

Public Member Functions

- **DataWriterQos** ()

  *Creates a **DataWriterQos** (p. 1226) with the default value for each policy.*
- **DataWriterQos** (const **dds::topic::qos::TopicQos** &topic_qos)

  *Creates a **DataWriterQos** (p. 1226) with the policies of a given TopicQos.*
- **DataWriterQos** & **operator=** (const **dds::topic::qos::TopicQos** &topic_qos)

  *Copies into this **DataWriterQos** (p. 1226) those policies that are also in TopicQos.*
- template<typename Policy >
  const Policy & **policy** () const

  Gets a QoS policy by const reference.
- template<typename Policy >
  Policy & **policy** ()

  Gets a QoS policy by reference.
- template<typename Policy >
  DataWriterQos & **policy** (const Policy &p)

  Sets a policy.
- template<typename Policy >
  DataWriterQos & **operator**<< (const Policy &p)

  Sets a policy.
- template<typename Policy >
  const **DataWriterQos** & **operator**>> (Policy &p) const

  Copies the values of a policy.

Related Functions

(Note that these are not member functions.)

- std::string **to_string** (const **DataWriterQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the **dds::pub::qos::DataWriterQos** (p. 1226)*
- std::string **to_string** (const **DataWriterQos** &qos, const **DataWriterQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the **dds::pub::qos::DataWriterQos** (p. 1226)*
- std::string **to_string** (const **DataWriterQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the dds::sub::qos::DataWriterQos*
- std::ostream & **operator**<< (std::ostream &out, const rti::pub::qos::DataWriterQos &qos)

  <<***extension***>> *(p. 354) Prints a **dds::pub::qos::DataWriterQos** (p. 1226) to an output stream.*

8.72.1 Detailed Description

<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::pub::DataWriter** (p. 1138) supports

8.72.2 DataWriterQos policies

A **DataWriterQos** (p. 1226) contains the following policies:

- **dds::core::policy::Durability** (p. 1433),
- **dds::core::policy::DurabilityService** (p. 1443),
- **dds::core::policy::Deadline** (p. 1255),
- **dds::core::policy::LatencyBudget** (p. 1631),
- **dds::core::policy::Liveliness** (p. 1645),
- **dds::core::policy::Reliability** (p. 2149),
- **dds::core::policy::DestinationOrder** (p. 1258),
- **dds::core::policy::History** (p. 1602),
- **dds::core::policy::ResourceLimits** (p. 2203),
- **dds::core::policy::TransportPriority** (p. 2552),
- **dds::core::policy::Lifespan** (p. 1635),
- **dds::core::policy::UserData** (p. 2591),
- **dds::core::policy::Ownership** (p. 1899),
- **dds::core::policy::OwnershipStrength** (p. 1906),
- **dds::core::policy::WriterDataLifecycle** (p. 2660),
- **dds::core::policy::DataRepresentation** (p. 1113),
- **dds::core::policy::DataTag** (p. 1132),
- **rti::core::policy::DataWriterResourceLimits** (p. 1235),
- **rti::core::policy::DataWriterProtocol** (p. 1210),
- **rti::core::policy::TransportSelection** (p. 2554),
- **rti::core::policy::TransportUnicast** (p. 2556),
- rti::core::policy::TransportEncapsulation,
- **rti::core::policy::PublishMode** (p. 2013),
- **rti::core::policy::Property** (p. 1969),
- **rti::core::policy::Service** (p. 2350),
- **rti::core::policy::Batch** (p. 880),
- **rti::core::policy::MultiChannel** (p. 1749),
- **rti::core::policy::Availability** (p. 868),
- **rti::core::policy::EntityName** (p. 1530),
- **rti::core::policy::TopicQueryDispatch** (p. 2522),

- **rti::core::policy::DataWriterTransferMode** (p. 1253),
- **rti::core::policy::TypeSupport** (p. 2574)

To get or set policies use the **policy()** (p. 1231) getters and setters or operator << (see **examples** (p. 603)).

You must set certain members in a consistent manner:

- **dds::core::policy::History** (p. 1602) .depth <= **dds::core::policy::ResourceLimits** (p. 2203) .max_samples↩ _per_instance
- **dds::core::policy::ResourceLimits** (p. 2203) .max_samples_per_instance <= **dds::core::policy::Resource↩ Limits** (p. 2203) .max_samples
- **dds::core::policy::ResourceLimits** (p. 2203) .initial_samples <= **dds::core::policy::ResourceLimits** (p. 2203) .max_samples
- **dds::core::policy::ResourceLimits** (p. 2203) .initial_instances <= **dds::core::policy::ResourceLimits** (p. 2203) .max_instances
- length of **dds::core::policy::UserData** (p. 2591) .value <= dds::domain::qos::DomainParticipantQos::resource↩ _limits .writer_user_data_max_length

If any of the above are not true, **dds::pub::DataWriter::qos(const dds::pub::qos::DataWriterQos&)** (p. 1166) and **dds::pub::Publisher::default_datawriter_qos(const dds::pub::qos::DataWriterQos&)** (p. 1998) will fail with **dds↩ ::core::InconsistentPolicyError** (p. 1610) and the **dds::pub::DataWriter** (p. 1138) constructors will fail with **dds↩ ::core::Error** (p. 1539)

Entity:

dds::pub::DataWriter (p. 1138)

See also

All QoS Policies (p. 508) allowed ranges within each Qos.

Qos Use Cases (p. 602)

Examples

Foo_publisher.cxx.

8.72.3 Constructor & Destructor Documentation

8.72.3.1 DataWriterQos() [1/2]

```
dds::pub::qos::DataWriterQos::DataWriterQos ( )
```

Creates a **DataWriterQos** (p. 1226) with the default value for each policy.

Note

If you configure Qos in XML, obtain the default value from the default QosProvider, since it can be overriden by configuration:
```
DataWriterQos writer_qos = dds::core::QosProvider::Default().datawriter_qos();
```

8.72.3.2 DataWriterQos() [2/2]

```
dds::pub::qos::DataWriterQos::DataWriterQos (
            const  dds::topic::qos::TopicQos & topic_qos )
```

Creates a **DataWriterQos** (p. 1226) with the policies of a given TopicQos.

The QoS policies that are exclusive to a **DataWriter** (p. 1138) retain their default values. A **dds::topic::qos::TopicQos** (p. 2509) contains a subset of the policies of a **DataWriterQos** (p. 1226). This constructor copies those common policies while initializing the policies that are only defined in **DataWriterQos** (p. 1226) to their default values.

Parameters

topic_qos	The TopicQos being copied.

Example:
```
DataWriterQos writer_qos(topic.qos());
```

8.72.4 Member Function Documentation

8.72.4.1 operator=()

```
DataWriterQos & dds::pub::qos::DataWriterQos::operator= (
            const  dds::topic::qos::TopicQos & topic_qos )  [inline]
```

Copies into this **DataWriterQos** (p. 1226) those policies that are also in TopicQos.

A **dds::topic::qos::TopicQos** (p. 2509) contains a subset of the policies of a **DataWriterQos** (p. 1226). This assignment operator copies those common policies into this instance while leaving the policies that are only defined in **DataWriterQos** (p. 1226) unaltered.

Parameters

topic_qos	The TopicQos to copy the common policies from.

For example:

```
// Load a TopicQos from a Qos profile
TopicQos topic_qos = dds::core::QosProvider::Default().topic_qos("MyLibrary::MyProfile");
// Create a DataWriterQos with the default policies
DataWriterQos writer_qos = dds::core::QosProvider::Default().datawriter_qos();
// Overwrite the policies that are also defined in topic_qos
writer_qos = topic_qos;
```

8.72.4.2 policy() [1/3]

```
template<typename Policy >
const Policy & dds::pub::qos::DataWriterQos::policy ( ) const
```

Gets a QoS policy by const reference.

Template Parameters

Policy	One of the **DataWriterQos policies** (p. 1228)

See also

Setting Qos Values (p. 603)

8.72.4.3 policy() [2/3]

```
template<typename Policy >
Policy & dds::pub::qos::DataWriterQos::policy ( )
```

Gets a QoS policy by reference.

Template Parameters

Policy	One of the **DataWriterQos policies** (p. 1228)

See also

Setting Qos Values (p. 603)

8.72.4.4 policy() [3/3]

```
template<typename Policy >
DataWriterQos & dds::pub::qos::DataWriterQos::policy (
            const Policy & p )  [inline]
```

Sets a policy.

See also

policy() (p. 1231)

Setting Qos Values (p. 603)

8.72.4.5 operator<<()

```
template<typename Policy >
DataWriterQos & dds::pub::qos::DataWriterQos::operator<< (
            const Policy & p )  [inline]
```

Sets a policy.

See also

policy() (p. 1231)

Setting Qos Values (p. 603)

8.72.4.6 operator>>()

```
template<typename Policy >
const  DataWriterQos & dds::pub::qos::DataWriterQos::operator>> (
            Policy & p ) const  [inline]
```

Copies the values of a policy.

Parameters

p	The destination where to copy the current value of the Policy

See also

policy() (p. 1231)

Setting Qos Values (p. 603)

8.72.5 Friends And Related Function Documentation

8.72.5.1 to_string() [1/3]

```
std::string to_string (
            const  DataWriterQos & qos,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::pub::qos::DataWriterQos** (p. 1226)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
DataWriterQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for DataWriterQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
DataWriterQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

8.72.5.2 to_string() [2/3]

```
std::string to_string (
            const  DataWriterQos & qos,
            const  DataWriterQos & base,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::pub::qos::DataWriterQos** (p. 1226)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

8.72.5.3 to_string() [3/3]

```
std::string to_string (
            const  DataWriterQos & qos,
            const  rti::core::qos_print_all_t & qos_print_all,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the dds::sub::qos::DataWriterQos

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
`to_string(qos, rti::core::qos_print_all);`

Returns

The string representation of the qos

8.72.5.4 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const rti::pub::qos::DataWriterQos & qos )  [related]
```

<<***extension***>> *(*p. *354)* Prints a **dds::pub::qos::DataWriterQos** (p. 1226) to an output stream.

8.73 rti::core::policy::DataWriterResourceLimits Class Reference

<<***extension***>> *(*p. *116)* Configures the memory usage of a **dds::pub::DataWriter** (p. 1138)

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **DataWriterResourceLimits** ()

  *Creates a **DataWriterResourceLimits** (*p. *1235) qos policy with default values.*
- **DataWriterResourceLimits** & **initial_concurrent_blocking_threads** (int32_t the_initial_concurrent_↩ blocking_threads)

  [DEPRECATED] (p. ***296)*** <<***extension***>> (p. *116) The initial number of threads that are allowed to concurrently block on write call on the same **dds::pub::DataWriter** (*p. *1138).*
- int32_t **initial_concurrent_blocking_threads** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **max_concurrent_blocking_threads** (int32_t the_max_concurrent_blocking_↩ threads)

  [DEPRECATED] (p. ***296)*** <<***extension***>> (p. *116) The maximum number of threads that are allowed to concurrently block on write call on the same **dds::pub::DataWriter** (*p. *1138).*
- int32_t **max_concurrent_blocking_threads** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **max_remote_reader_filters** (int32_t the_max_remote_reader_filters)

  <<***extension***>> (p. *116) The maximum number of remote DataReaders for which the **dds::pub::DataWriter** (*p. *1138) will perform content-based filtering.*
- int32_t **max_remote_reader_filters** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **initial_batches** (int32_t the_initial_batches)

  <<***extension***>> (p. *116) Represents the initial number of batches a **dds::pub::DataWriter** (*p. *1138) will manage.*
- int32_t **initial_batches** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **max_batches** (int32_t the_max_batches)

  <<***extension***>> (p. *116) Represents the maximum number of batches a **dds::pub::DataWriter** (*p. *1138) will manage.*
- int32_t **max_batches** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **cookie_max_length** (int32_t max_length)

  *Represents the maximum length in bytes of a **rti::core::Cookie** (*p. *965).*
- int32_t **cookie_max_length** () const

Getter (see setter with the same name)

- **DataWriterResourceLimits** & **instance_replacement** (**DataWriterResourceLimitsInstanceReplacement↩ Kind** the_instance_replacement)

  *<<**extension**>> (p. 116) Sets the kinds of instances allowed to be replaced when instance resource limits are reached.*
- **DataWriterResourceLimitsInstanceReplacementKind** **instance_replacement** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **replace_empty_instances** (bool the_replace_empty_instances)

  *<<**extension**>> (p. 116) Whether or not to replace empty instances during instance replacement*
- bool **replace_empty_instances** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **autoregister_instances** (bool the_autoregister_instances)

  ***[DEPRECATED] (**p. **296)** <<**extension**>> (p. 116) Whether or not to automatically register new instances*
- bool **autoregister_instances** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **initial_virtual_writers** (int32_t the_initial_virtual_writers)

  *<<**extension**>> (p. 116) The initial number of virtual writers supported by a **dds::pub::DataWriter** (p. 1138)*
- int32_t **initial_virtual_writers** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **max_virtual_writers** (int32_t the_max_virtual_writers)

  *<<**extension**>> (p. 116) The maximum number of virtual writers supported by a **dds::pub::DataWriter** (p. 1138)*
- int32_t **max_virtual_writers** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **initial_remote_readers** (int32_t the_initial_remote_readers)

  *<<**extension**>> (p. 116) Controls the initial memory allocation for the matched remote readers of a **dds::pub::Data↩ Writer** (p. 1138).*
- **DataWriterResourceLimits** & **max_remote_readers** (int32_t the_max_remote_readers)

  *<<**extension**>> (p. 116) The maximum number of remote readers supported by a **dds::pub::DataWriter** (p. 1138)*
- int32_t **initial_remote_readers** () const

  Getter (see setter with the same name)
- int32_t **max_remote_readers** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **max_app_ack_remote_readers** (int32_t the_max_app_ack_remote_readers)

  *<<**extension**>> (p. 116) The maximum number of application-level acknowledging remote readers supported by a **dds::pub::DataWriter** (p. 1138)*
- int32_t **max_app_ack_remote_readers** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **initial_active_topic_queries** (int32_t the_initial_active_topic_queries)

  *<<**extension**>> (p. 116) Represents the initial number of active topic queries a **dds::pub::DataWriter** (p. 1138) will manage.*
- int32_t **initial_active_topic_queries** () const

  Getter (see setter with the same name)
- **DataWriterResourceLimits** & **max_active_topic_queries** (int32_t the_max_active_topic_queries)

  *<<**extension**>> (p. 116) Represents the maximum number of active topic queries a **dds::pub::DataWriter** (p. 1138) will manage.*
- int32_t **max_active_topic_queries** () const

  Getter (see setter with the same name)
- const **AllocationSettings** & **writer_loaned_sample_allocation** () const

*<<**extension**>> (p. 116) Represents the allocation settings of loaned samples managed by a **dds::pub::DataWriter** (p. 1138).*

- **AllocationSettings** & **writer_loaned_sample_allocation** ()

  Getter by non-const reference (see getter by const reference with the same name)

- **DataWriterResourceLimits** & **initialize_writer_loaned_sample** (bool the_initialize_writer_loaned_sample)

  *<<**extension**>> (p. 116) Whether or not to initialize loaned samples returned by a **dds::pub::DataWriter** (p. 1138)*

- bool **initialize_writer_loaned_sample** () const

  Getter (see setter with the same name)

8.73.1 Detailed Description

*<<**extension**>> (p. 116)* Configures the memory usage of a **dds::pub::DataWriter** (p. 1138)

DataWriters must allocate internal structures to handle the simultaneously blocking of threads trying to call **dds::pub::DataWriter::write()** (p. 1148) on the same **dds::pub::DataWriter** (p. 1138), for the storage used to batch small samples, and for content-based filters specified by DataReaders.

Most of these internal structures start at an initial size and, by default, will be grown as needed by dynamically allocating additional memory. You may set fixed, maximum sizes for these internal structures if you want to bound the amount of memory that can be used by a **dds::pub::DataWriter** (p. 1138). By setting the initial size to the maximum size, you will prevent RTI Connext from dynamically allocating any memory after the creation of the **dds::pub::DataWriter** (p. 1138).

This QoS policy is an extension to the DDS standard.

Entity:

dds::pub::DataWriter (p. 1138)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.73.2 Constructor & Destructor Documentation

8.73.2.1 DataWriterResourceLimits()

```
rti::core::policy::DataWriterResourceLimits::DataWriterResourceLimits ( )  [inline]
```

Creates a **DataWriterResourceLimits** (p. 1235) qos policy with default values.

8.73.3 Member Function Documentation

8.73.3.1 initial_concurrent_blocking_threads() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::initial_concurrent_↩
blocking_threads (
            int32_t the_initial_concurrent_blocking_threads )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The initial number of threads that are allowed to concurrently block on write call on the same **dds::pub::DataWriter** (p. 1138).

This value only applies if **dds::core::policy::History** (p. 1602) has its kind set to **dds::core::policy::HistoryKind_↩ def::KEEP_ALL** (p. 1607) and **dds::core::policy::Reliability::max_blocking_time** (p. 2153) is $>$ 0.

[default] 1

[range] [1, 10000], <= max_concurrent_blocking_threads

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Concurrency** (p. 164)

8.73.3.2 initial_concurrent_blocking_threads() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::initial_concurrent_blocking_threads ( ) const
```

Getter (see setter with the same name)

8.73.3.3 max_concurrent_blocking_threads() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::max_concurrent_blocking↩
_threads (
            int32_t the_max_concurrent_blocking_threads )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum number of threads that are allowed to concurrently block on write call on the same **dds::pub::DataWriter** (p. 1138).

This value only applies if **dds::core::policy::History** (p. 1602) has its kind set to **dds::core::policy::HistoryKind_↩ def::KEEP_ALL** (p. 1607) and **dds::core::policy::Reliability::max_blocking_time** (p. 2153) is $>$ 0.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 10000] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_concurrent_blocking_threads

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Concurrency** (p. 164)

8.73.3.4 max_concurrent_blocking_threads() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::max_concurrent_blocking_threads ( ) const
```

Getter (see setter with the same name)

8.73.3.5 max_remote_reader_filters() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::max_remote_reader_filters
(
            int32_t the_max_remote_reader_filters )
```

<<***extension***>> *(*p. *116)* The maximum number of remote DataReaders for which the **dds::pub::DataWriter** (p. 1138) will perform content-based filtering.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [0, (2^31)-2] or **dds::core::LENGTH_UNLIMITED** (p. 438).

0: The **dds::pub::DataWriter** (p. 1138) will not perform filtering for any **dds::sub::DataReader** (p. 976).

1 to (2^31)-2: The DataWriter will filter for up to the specified number of DataReaders. In addition, the Datawriter will store the result of the filtering per sample per DataReader.

dds::core::LENGTH_UNLIMITED (p. 438): The DataWriter will filter for up to (2^31)-2 DataReaders. However, in this case, the DataWriter will not store the filtering result per sample per DataReader. Thus, if a sample is resent (such as due to a loss of reliable communication), the sample will be filtered again.

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Content Filtering** (p. 165)

8.73.3.6 max_remote_reader_filters() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::max_remote_reader_filters ( ) const
```

Getter (see setter with the same name)

8.73.3.7 initial_batches() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::initial_batches (
            int32_t the_initial_batches )
```

<<***extension***>> *(*p. *116)* Represents the initial number of batches a **dds::pub::DataWriter** (p. 1138) will manage.

[default] 8

[range] [1,100 million]

See also

rti::core::policy::Batch (p. 880)

Categories

Immutable (p. 243), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Batching** (p. 161)

8.73.3.8 initial_batches() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::initial_batches ( ) const
```

Getter (see setter with the same name)

8.73.3.9 max_batches() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::max_batches (
            int32_t the_max_batches )
```

<<***extension***>> *(*p. *116)* Represents the maximum number of batches a **dds::pub::DataWriter** (p. 1138) will manage.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

When batching is enabled, the maximum number of samples that a **dds::pub::DataWriter** (p. 1138) can store is limited by this value and **dds::core::policy::ResourceLimits::max_samples** (p. 2205).

[range] [1,100 million] or **dds::core::LENGTH_UNLIMITED** (p. 438) >= DDS_RtpsReliableWriterProtocol_t↩ ::heartbeats_per_max_samples if batching is enabled

See also

rti::core::policy::Batch (p. 880)

Categories

Immutable (p. 243), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Batching** (p. 161)

8.73.3.10 max_batches() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::max_batches ( ) const
```

Getter (see setter with the same name)

8.73.3.11 cookie_max_length() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::cookie_max_length (
            int32_t max_length )
```

Represents the maximum length in bytes of a **rti::core::Cookie** (p. 965).

[default] dds::core::LENGTH_UNLIMITED (p. 438)

Sets the maximum allowed byte-sequence length of a **rti::core::Cookie** (p. 965) used when writing with parameters

[range] [1,100 million] or **dds::core::LENGTH_UNLIMITED** (p. 438)

See also

dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&) (p. 1179), **dds::pub::DataWriter::dispose_instance(rti::pub::WriteParams&)** (p. 1173), **dds::pub::DataWriter::register_instance(const T&, rti::pub::WriteParams&)** (p. 1185), **dds::pub::DataWriter::unregister_instance(rti::pub::WriteParams&)** (p. 1173)

8.73.3.12 cookie_max_length() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::cookie_max_length ( ) const
```

Getter (see setter with the same name)

8.73.3.13 instance_replacement() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::instance_replacement (
            DataWriterResourceLimitsInstanceReplacementKind the_instance_replacement )
```

<<***extension***>> *(p. 116)* Sets the kinds of instances allowed to be replaced when instance resource limits are reached.

When a **dds::pub::DataWriter** (p. 1138)'s number of active instances is greater than **dds::core::policy::ResourceLimits::max_instances** (p. 2206), it will try to make room by replacing an existing instance. This field specifies the kinds of instances allowed to be replaced.

If a replaceable instance is not available, either an out-of-resources exception will be returned, or the writer may block if the instance reclamation was done when writing.

[default] DataWriterResourceLimitsInstanceReplacementKind::UNREGISTERED_INSTANCE_REPLACEMENT

See also

DataWriterResourceLimitsInstanceReplacementKind

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Instance Lifecycle** (p. 193)

8.73.3.14 instance_replacement() [2/2]

```
DataWriterResourceLimitsInstanceReplacementKind rti::core::policy::DataWriterResourceLimits↩
::instance_replacement ( ) const
```

Getter (see setter with the same name)

8.73.3.15 replace_empty_instances() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::replace_empty_instances (
            bool the_replace_empty_instances )
```

<<***extension***>> *(*p. *116)* Whether or not to replace empty instances during instance replacement

When a **dds::pub::DataWriter** (p. 1138) has more active instances than allowed by **dds::core::policy::Resource↩ Limits::max_instances** (p. 2206), it tries to make room by replacing an existing instance. This field configures whether empty instances (i.e. instances with no samples) may be replaced. If set true, then a **dds::pub::DataWriter** (p. 1138) will first try reclaiming empty instances, before trying to replace whatever is specified by **rti::core::policy::DataWriter↩ ResourceLimits::instance_replacement** (p. 1241).

[default] false

See also

DataWriterResourceLimitsInstanceReplacementKind

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Instance Lifecycle** (p. 193)

8.73.3.16 replace_empty_instances() [2/2]

```
bool rti::core::policy::DataWriterResourceLimits::replace_empty_instances ( ) const
```

Getter (see setter with the same name)

8.73.3.17 autoregister_instances() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::autoregister_instances (
            bool the_autoregister_instances )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Whether or not to automatically register new instances

[default] false

When set to true, it is possible to write with a non-NIL handle of an instance that is not registered: the write operation will succeed and the instance will be registered. Otherwise, that write operation would fail.

See also

dds::pub::DataWriter::write() (p. 1148)

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193)

8.73.3.18 autoregister_instances() [2/2]

```
bool rti::core::policy::DataWriterResourceLimits::autoregister_instances ( ) const
```

Getter (see setter with the same name)

8.73.3.19 initial_virtual_writers() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::initial_virtual_writers (
            int32_t the_initial_virtual_writers )
```

<<***extension***>> *(p. 116)* The initial number of virtual writers supported by a **dds::pub::DataWriter** (p. 1138)

[default] 1

[range] [1, 1000000], or **dds::core::LENGTH_UNLIMITED** (p. 438)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.73.3.20 initial_virtual_writers() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::initial_virtual_writers ( ) const
```

Getter (see setter with the same name)

8.73.3.21 max_virtual_writers() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::max_virtual_writers (
            int32_t the_max_virtual_writers )
```

<<***extension***>> *(*p. *116)* The maximum number of virtual writers supported by a **dds::pub::DataWriter** (p. 1138)

Sets the maximum number of unique virtual writers supported by a **dds::pub::DataWriter** (p. 1138), where virtual writers are added when samples are written with the virtual writer GUID.

This field is specially relevant in the configuration of Persistence **Service** (p. 2350) DataWriters since these DataWriters will publish samples on behalf of multiple virtual writers.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1000000], or **dds::core::LENGTH_UNLIMITED** (p. 438)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.73.3.22 max_virtual_writers() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::max_virtual_writers ( ) const
```

Getter (see setter with the same name)

8.73.3.23 initial_remote_readers() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::initial_remote_readers (
            int32_t the_initial_remote_readers )
```

<<***extension***>> *(*p. *116)* Controls the initial memory allocation for the matched remote readers of a **dds::pub::Data↩ Writer** (p. 1138).

The larger the value, the more memory is pre-allocated, the less memory is dynamically allocated.

[default] 2

[range] [0, 1000000], <= max_remote_readers

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.73.3.24 max_remote_readers() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::max_remote_readers (
            int32_t the_max_remote_readers )
```

<<***extension***>> *(p. 116)* The maximum number of remote readers supported by a **dds::pub::DataWriter** (p. 1138)

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1000000], or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_remote_readers

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.73.3.25 initial_remote_readers() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::initial_remote_readers ( ) const
```

Getter (see setter with the same name)

8.73.3.26 max_remote_readers() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::max_remote_readers ( ) const
```

Getter (see setter with the same name)

8.73.3.27 max_app_ack_remote_readers() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::max_app_ack_remote_↩
readers (
            int32_t the_max_app_ack_remote_readers )
```

<<***extension***>> *(p. 116)* The maximum number of application-level acknowledging remote readers supported by a **dds::pub::DataWriter** (p. 1138)

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1000000], or **dds::core::LENGTH_UNLIMITED** (p. 438)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application Acknowledgement** (p. 158), **Reliability** (p. 222)

8.73.3.28 max_app_ack_remote_readers() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::max_app_ack_remote_readers ( ) const
```

Getter (see setter with the same name)

8.73.3.29 initial_active_topic_queries() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::initial_active_topic_↩
queries (
            int32_t the_initial_active_topic_queries )
```

<<***extension***>> *(p. 116)* Represents the initial number of active topic queries a **dds::pub::DataWriter** (p. 1138) will manage.

[default] 1

[range] [1, 1000000]

See also

rti::core::policy::TopicQueryDispatch (p. 2522)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Topic Queries** (p. 239)

8.73.3.30 initial_active_topic_queries() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::initial_active_topic_queries ( ) const
```

Getter (see setter with the same name)

8.73.3.31 max_active_topic_queries() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::max_active_topic_queries
(
            int32_t the_max_active_topic_queries )
```

<<***extension***>> *(*p. *116)* Represents the maximum number of active topic queries a **dds::pub::DataWriter** (p. 1138) will manage.

When topic queries are enabled, the maximum number of topic queries that a **dds::pub::DataWriter** (p. 1138) can publish data samples for at the same time is limited by this value.

When the DataWriter receives one topic query above this limit, it will wait to process it until it finishes publishing all the samples for at least one of the current topic queries.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1000000] or **dds::core::LENGTH_UNLIMITED** (p. 438)

See also

rti::core::policy::TopicQueryDispatch (p. 2522)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Topic Queries** (p. 239)

8.73.3.32 max_active_topic_queries() [2/2]

```
int32_t rti::core::policy::DataWriterResourceLimits::max_active_topic_queries ( ) const
```

Getter (see setter with the same name)

8.73.3.33 writer_loaned_sample_allocation() [1/2]

```
const AllocationSettings & rti::core::policy::DataWriterResourceLimits::writer_loaned_sample_↩
allocation ( ) const
```

<<***extension***>> *(*p. *116)* Represents the allocation settings of loaned samples managed by a **dds::pub::DataWriter** (p. 1138).

The number of samples loaned by a **dds::pub::DataWriter** (p. 1138) via **dds::pub::DataWriter::get_loan** (p. 1183) is limited by the **rti::core::AllocationSettings::max_count** (p. 804) of **rti::core::policy::DataWriterResourceLimits↩ ::writer_loaned_sample_allocation** (p. 1247). **dds::pub::DataWriter::get_loan** (p. 1183) returns NULL if and only if **rti::core::AllocationSettings::max_count** (p. 804) samples have been loaned, and none of those samples has been written with **dds::pub::DataWriter::write()** (p. 1148) or discarded via **dds::pub::DataWriter::discard_loan** (p. 1185).

[default] initial_count = **rti::core::AllocationSettings::AUTO_COUNT** (p. 805) (**dds::core::policy::Resource↩ Limits::initial_samples** (p. 2207) + 1); max_count = **rti::core::AllocationSettings::AUTO_COUNT** (p. 805) (**dds↩ ::core::policy::ResourceLimits::max_samples** (p. 2205) + 1); incremental_count = **rti::core::AllocationSettings↩ ::AUTO_COUNT** (p. 805) (0 if initial_count = max_count; initial_count otherwise);

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

See also

dds::pub::DataWriter::get_loan (p. 1183)

dds::pub::DataWriter::discard_loan (p. 1185)

Categories

Immutable (p. 243), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Sending Large Data** (p. 197)

8.73.3.34 writer_loaned_sample_allocation() [2/2]

```
AllocationSettings & rti::core::policy::DataWriterResourceLimits::writer_loaned_sample_allocation
( )
```

Getter by non-const reference (see getter by const reference with the same name)

8.73.3.35 initialize_writer_loaned_sample() [1/2]

```
DataWriterResourceLimits & rti::core::policy::DataWriterResourceLimits::initialize_writer_↩
loaned_sample (
            bool the_initialize_writer_loaned_sample )
```

<<***extension***>> *(p. 116)* Whether or not to initialize loaned samples returned by a **dds::pub::DataWriter** (p. 1138)

[default] false

See also

dds::pub::DataWriter::get_loan (p. 1183)

Categories

Immutable (p. 243), **Sample Memory Management** (p. 230), **Sending Large Data** (p. 197)

8.73.3.36 initialize_writer_loaned_sample() [2/2]

```
bool rti::core::policy::DataWriterResourceLimits::initialize_writer_loaned_sample ( ) const
```

Getter (see setter with the same name)

8.74 rti::core::policy::DataWriterResourceLimitsInstanceReplacement↩ Kind_def Struct Reference

<<***extension***>> *(p. 354)* The enumeration for DataWriter Resource Limits

```
#include <PolicyKind.hpp>
```

Public Types

- enum **type** {
  UNREGISTERED ,
  ALIVE ,
  DISPOSED ,
  ALIVE_THEN_DISPOSED ,
  DISPOSED_THEN_ALIVE ,
  ALIVE_OR_DISPOSED }

  The underlying `enum` *type.*

8.74.1 Detailed Description

<<***extension***>> *(p. 354)* The enumeration for DataWriter Resource Limits

When **dds::core::policy::ResourceLimits::max_instances** (p. 2206) is reached, a **dds::pub::DataWriter** (p. 1138) will try to make room for a new instance by attempting to reclaim an existing instance based on the instance replacement kind specified by **rti::core::policy::DataWriterResourceLimits::instance_replacement** (p. 1241).

Only instances whose states match the specified kinds are eligible to be replaced. In addition, an instance must have had all of its samples fully acknowledged for it to be considered replaceable.

For all kinds, a **dds::pub::DataWriter** (p. 1138) will replace the oldest instance satisfying that kind. For example, when the kind is DataWriterResourceLimitsInstanceReplacementKind::UNREGISTERED_INSTANCE_REPLACEMENT, a **dds::pub::DataWriter** (p. 1138) will remove the oldest, fully acknowledged, unregistered instance, if such an instance exists.

If no replaceable instance exists, the invoked function will either return with an appropriate out-of-resources return code, or in the case of a write, it may first block to wait for an instance to be acknowledged. Otherwise, the **dds::pub::↩ DataWriter** (p. 1138) will replace the old instance with the new instance, and invoke, if available, the **dds::pub::Data↩ WriterListener::on_instance_replaced** (p. 1206) to notify the user about an instance being replaced.

A **dds::pub::DataWriter** (p. 1138) checks for replaceable instances in the following order, stopping once a replaceable instance is found:

- If **rti::core::policy::DataWriterResourceLimits::replace_empty_instances** (p. 1242) is true, a **dds::pub::↩ DataWriter** (p. 1138) first tries replacing instances that have no samples. These empty instances can be unregistered, disposed, or alive.
- Next, a **dds::pub::DataWriter** (p. 1138) tries replacing unregistered instances. Since an unregistered instance indicates that the **dds::pub::DataWriter** (p. 1138) is done modifying it, unregistered instances are replaced before instances of any other state (alive, disposed). This is the same as the DataWriterResourceLimitsInstance↩ ReplacementKind::UNREGISTERED_INSTANCE_REPLACEMENT kind.
- Then, a **dds::pub::DataWriter** (p. 1138) tries replacing what is specified by **rti::core::policy::DataWriter↩ ResourceLimits::instance_replacement** (p. 1241). With unregistered instances already checked, this leaves alive and disposed instances. When both alive and disposed instances may be replaced, the kind specifies whether the particular order matters (e.g., DISPOSED_THEN_ALIVE, ALIVE_THEN_DISPOSED) or not (ALIVE_OR_DISPOSED).

QoS:

rti::core::policy::DataWriterResourceLimits (p. 1235)

8.74.2 Member Enumeration Documentation

8.74.2.1 type

`enum` **`rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_def::type`**

The underlying `enum` type.

Enumerator

UNREGISTERED	Allows a **dds::pub::DataWriter** (p. 1138) to reclaim unregistered acknowledged instances. By default, all instance replacement kinds first attempt to reclaim an unregistered, acknowledged instance. Used in **rti::core::policy::DataWriterResourceLimits::instance_replacement** (p. 1241) **[default]**
ALIVE	Allows a **dds::pub::DataWriter** (p. 1138) to reclaim alive, acknowledged instances. When an unregistered, acknowledged instance is not available to reclaim, this kind allows a **dds::pub::DataWriter** (p. 1138) to reclaim an alive, acknowledged instance, where an alive instance is a registered, non-disposed instance. The least recently registered or written alive instance will be reclaimed.
DISPOSED	Allows a **dds::pub::DataWriter** (p. 1138) to reclaim disposed acknowledged instances. When an unregistered, acknowledged instance is not available to reclaim, this kind allows a **dds::pub::DataWriter** (p. 1138) to reclaim a disposed, acknowledged instance. The least recently disposed instance will be reclaimed.
ALIVE_THEN_DISPOSED	Allows a **dds::pub::DataWriter** (p. 1138) first to reclaim an alive, acknowledged instance, and then, if necessary, a disposed, acknowledged instance. When an unregistered, acknowledged instance is not available to reclaim, this kind allows a **dds::pub::DataWriter** (p. 1138) to first try reclaiming an alive, acknowledged instance. If no instance is reclaimable, then it tries reclaiming a disposed, acknowledged instance. The least recently used (i.e., registered, written, or disposed) instance will be reclaimed.
DISPOSED_THEN_ALIVE	Allows a **dds::pub::DataWriter** (p. 1138) first to reclaim a disposed, acknowledged instance, and then, if necessary, an alive, acknowledged instance. When an unregistered, acknowledged instance is not available to reclaim, this kind allows a **dds::pub::DataWriter** (p. 1138) to first try reclaiming a disposed, acknowledged instance. If no instance is reclaimable, then it tries reclaiming an alive, acknowledged instance. The least recently used (i.e., disposed, registered, or written) instance will be reclaimed.
ALIVE_OR_DISPOSED	Allows a **dds::pub::DataWriter** (p. 1138) to reclaim a either an alive acknowledged instance or a disposed acknowledged instance. When an unregistered acknowledged instance is not available to reclaim, this kind allows a **dds::pub::DataWriter** (p. 1138) to reclaim either an alive, acknowledged instance or a disposed, acknowledged instance. If both instance kinds are available to reclaim, the **dds::pub::DataWriter** (p. 1138) will reclaim the least recently used (i.e. disposed, registered, or written) instance.

8.75 rti::core::DataWriterShmemRefTransferModeSettings Class Reference

<<***extension***>> *(p. 354)* Configures aspects of the shared memory reference transfer mode related to a DataWriter

`#include <rti/core/PolicySettings.hpp>`

Public Member Functions

- **DataWriterShmemRefTransferModeSettings** ()

  Creates an instance with the default settings.

- **DataWriterShmemRefTransferModeSettings** & **enable_data_consistency_check** (bool the_enable_data_↩ consistency_check)

  Controls if samples can be checked for consistency.
- bool **enable_data_consistency_check** () const

  Getter (see setter with the same name)

8.75.1 Detailed Description

<<***extension***>> *(*p. *354)* Configures aspects of the shared memory reference transfer mode related to a DataWriter

It is used to configure a **dds::pub::DataWriter** (p. 1138) using **Zero Copy transfer over shared memory** (p. 50).

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

QoS:

rti::core::policy::DataWriterTransferMode (p. 1253)

8.75.2 Constructor & Destructor Documentation

8.75.2.1 DataWriterShmemRefTransferModeSettings()

```
rti::core::DataWriterShmemRefTransferModeSettings::DataWriterShmemRefTransferModeSettings ( )
[inline]
```

Creates an instance with the default settings.

8.75.3 Member Function Documentation

8.75.3.1 enable_data_consistency_check() [1/2]

```
DataWriterShmemRefTransferModeSettings & rti::core::DataWriterShmemRefTransferModeSettings↩
::enable_data_consistency_check (
            bool the_enable_data_consistency_check )
```

Controls if samples can be checked for consistency.

When this setting is true, the **dds::pub::DataWriter** (p. 1138) sends an incrementing sequence number as an inline QoS with every sample. This sequence number allows a **dds::sub::DataReader** (p. 976) to use the **dds::sub::Data↩ Reader::is_data_consistent** (p. 1022) API to detect if the **dds::pub::DataWriter** (p. 1138) overwrote the sample before the **dds::sub::DataReader** (p. 976) could complete processing the sample.

[default] true

8.75.3.2 enable_data_consistency_check() [2/2]

```
bool rti::core::DataWriterShmemRefTransferModeSettings::enable_data_consistency_check ( ) const
```

Getter (see setter with the same name)

8.76 rti::core::policy::DataWriterTransferMode Class Reference

<<***extension***>> *(*p. *116)* Configures the transfer mode of a **dds::pub::DataWriter** (p. 1138)

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **DataWriterTransferMode** ()

  *Creates a **DataWriterTransferMode** (*p. *1253) qos policy with default values.*
- **DataWriterTransferMode** & **shmem_ref_settings** (const **rti::core::DataWriterShmemRefTransferMode↩ Settings** &settings)

  <<***extension***>> *(*p. *116) Settings related to transferring data using shared memory references.*
- const **rti::core::DataWriterShmemRefTransferModeSettings** & **shmem_ref_settings** () const

  Getter by const reference (see setter)
- **rti::core::DataWriterShmemRefTransferModeSettings** & **shmem_ref_settings** ()

  Getter by reference (see setter)

Static Public Member Functions

- static **DataWriterTransferMode ShmemRefSettings** (bool enable_data_consistency_check)

  Creates a policy with ShmemRefSettings.

8.76.1 Detailed Description

<<***extension***>> *(p. 116)* Configures the transfer mode of a **dds::pub::DataWriter** (p. 1138)

It contains qualitative settings related to the actions a **dds::pub::DataWriter** (p. 1138) performs while transferring its data.

Entity:

dds::pub::DataWriter (p. 1138)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.76.2 Constructor & Destructor Documentation

8.76.2.1 DataWriterTransferMode()

```
rti::core::policy::DataWriterTransferMode::DataWriterTransferMode ( )  [inline]
```

Creates a **DataWriterTransferMode** (p. 1253) qos policy with default values.

8.76.3 Member Function Documentation

8.76.3.1 ShmemRefSettings()

```
static DataWriterTransferMode rti::core::policy::DataWriterTransferMode::ShmemRefSettings (
            bool enable_data_consistency_check )  [inline], [static]
```

Creates a policy with ShmemRefSettings.

8.76.3.2 shmem_ref_settings() [1/3]

```
DataWriterTransferMode & rti::core::policy::DataWriterTransferMode::shmem_ref_settings (
            const  rti::core::DataWriterShmemRefTransferModeSettings & settings )
```

<<***extension***>> *(p. 116)* Settings related to transferring data using shared memory references.

For details, refer to the **rti::core::DataWriterShmemRefTransferModeSettings** (p. 1251)

Categories

Immutable (p. 243), **Sending Large Data** (p. 197)

Referenced by **rti::core::policy::DataReaderResourceLimits::max_samples_per_remote_writer()**.

8.76.3.3 shmem_ref_settings() [2/3]

```
const  rti::core::DataWriterShmemRefTransferModeSettings & rti::core::policy::DataWriterTransfer↩
Mode::shmem_ref_settings ( ) const
```

Getter by const reference (see setter)

8.76.3.4 shmem_ref_settings() [3/3]

```
 rti::core::DataWriterShmemRefTransferModeSettings & rti::core::policy::DataWriterTransferMode↩
::shmem_ref_settings ( )
```

Getter by reference (see setter)

8.77 dds::core::policy::Deadline Class Reference

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **Deadline** ()
  Creates the default deadline, with an infinite period.
- **Deadline** (const **dds::core::Duration** &d)
  Creates a deadline policy with the specified period.
- **Deadline** & **period** (const **dds::core::Duration** &the_period)
  Sets the duration of the deadline period.
- const **dds::core::Duration period** () const
  Getter (see setter with the same name)

8.77.1 Detailed Description

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

A **dds::sub::DataReader** (p. 976) expects a new sample updating the value of each instance at least once every `period`. That is, `period` specifies the maximum expected elapsed time between arriving data samples.

A **dds::pub::DataWriter** (p. 1138) indicates that the application commits to write a new value (using the **dds::pub::DataWriter** (p. 1138)) for each instance managed by the **dds::pub::DataWriter** (p. 1138) at least once every `period`.

This QoS can be used during system integration to ensure that applications have been coded to meet design specifications.

It can also be used during runtime to detect when systems are performing outside of design specifications. Receiving applications can take appropriate actions to prevent total system failure when data is not received in time. For topics on which data is not expected to be periodic, `period` should be set to an infinite value.

Entity:

dds::topic::Topic (p. 2474), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Status:

dds::core::status::StatusMask::offered_deadline_missed() (p. 2379), **dds::core::status::StatusMask::requested_deadline_missed()** (p. 2380), **dds::core::status::StatusMask::offered_incompatible_qos()** (p. 2380), **dds::core::status::StatusMask::requested_incompatible_qos()** (p. 2381)

Properties:

RxO (p. 514) = YES
Changeable (p. 514) = **YES** (p. 514)

8.77.2 Usage

This policy is useful for cases where a **dds::topic::Topic** (p. 2474) is expected to have each instance updated periodically. On the publishing side this setting establishes a contract that the application must meet. On the subscribing side the setting establishes a minimum requirement for the remote publishers that are expected to supply the data values.

When RTI Connext 'matches' a **dds::pub::DataWriter** (p. 1138) and a **dds::sub::DataReader** (p. 976) it checks whether the settings are compatible (i.e., *offered deadline* $<=$ *requested deadline*); if they are not, the two entities are informed (via the **Listener** (p. 1637) or **dds::core::cond::Condition** (p. 948) mechanism) of the incompatibility of the QoS settings and communication will not occur.

Assuming that the reader and writer ends have compatible settings, the fulfilment of this contract is monitored by RTI Connext and the application is informed of any violations by means of the proper **Listener** (p. 1637) or **dds::core::cond::Condition** (p. 948).

8.77.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered period* <= *requested period* holds.

8.77.4 Consistency

The setting of the **DEADLINE** (p. 523) policy must be set consistently with that of the **TIME_BASED_FILTER** (p. 547).

For these two policies to be consistent the settings must be such that *deadline period* >= *minimum_separation*.

An attempt to set these policies in an inconsistent manner will result in **dds::core::InconsistentPolicyError** (p. 1610) in **set_qos (abstract)** (p. 1522), or the **dds::core::Entity** (p. 1520) will not be created.

For a **dds::sub::DataReader** (p. 976), the **DEADLINE** (p. 523) policy and **dds::core::policy::TimeBasedFilter** (p. 2469) may interact such that even though the **dds::pub::DataWriter** (p. 1138) is writing samples fast enough to fulfill its commitment to its own deadline, the **dds::sub::DataReader** (p. 976) may see violations of its deadline. This happens because RTI Connext will drop any samples received within the **dds::core::policy::TimeBasedFilter**↩ **::minimum_separation** (p. 2472). To avoid triggering the **dds::sub::DataReader** (p. 976)'s deadline, even though the matched **dds::pub::DataWriter** (p. 1138) is meeting its own deadline, set the two QoS parameters so that:

reader deadline >= *reader minimum_separation + writer deadline*

See **dds::core::policy::TimeBasedFilter** (p. 2469) for more information about the interactions between deadlines and time-based filters.

See also

dds::core::policy::TimeBasedFilter (p. 2469)

8.77.5 Constructor & Destructor Documentation

8.77.5.1 Deadline() [1/2]

```
dds::core::policy::Deadline::Deadline ( )  [inline]
```

Creates the default deadline, with an infinite period.

8.77.5.2 Deadline() [2/2]

```
dds::core::policy::Deadline::Deadline (
            const  dds::core::Duration & d )  [inline], [explicit]
```

Creates a deadline policy with the specified period.

8.77.6 Member Function Documentation

8.77.6.1 period() [1/2]

```
Deadline & dds::core::policy::Deadline::period (
            const  dds::core::Duration & the_period )  [inline]
```

Sets the duration of the deadline period.

[default] dds::core::Duration::infinite() (p. 1451)

[range] [1 nanosec, 1 year] or **dds::core::Duration::infinite()** (p. 1451), >= **dds::core::policy::TimeBasedFilter::minimum_separation** (p. 2472)

Categories

Mutable (p. 280), **Discovery Metadata** (p. 288), **Matching** (p. 293), **Applied Per-Instance** (p. 287), **Data Availability** (p. 167)

8.77.6.2 period() [2/2]

```
const  dds::core::Duration dds::core::policy::Deadline::period ( ) const  [inline]
```

Getter (see setter with the same name)

8.78 dds::core::policy::DestinationOrder Class Reference

Controls the logical order of updates to the same instance by a **dds::pub::Publisher** (p. 1993).

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **DestinationOrder** ()
  Creates the default policy.
- **DestinationOrder** (**dds::core::policy::DestinationOrderKind** the_kind)
  Creates a policy with the specified destination order kind.
- **DestinationOrder** & **kind** (**dds::core::policy::DestinationOrderKind** the_kind)
  Sets the destination order kind.
- **dds::core::policy::DestinationOrderKind kind** () const
  Getter (see the setter with the same name)
- **dds::core::policy::DestinationOrder** & **scope** (**rti::core::policy::DestinationOrderScopeKind** the_scope)
  *<<**extension**>> (p. 116) Sets the destination order scope*
- **rti::core::policy::DestinationOrderScopeKind scope** () const
  *<<**extension**>> (p. 116) Getter (see the setter with the same name)*
- **dds::core::policy::DestinationOrder** & **source_timestamp_tolerance** (const **dds::core::Duration** &ms)
  *<<**extension**>> (p. 116) Sets the allowed tolerance between source timestamps of consecutive samples.*
- **dds::core::Duration source_timestamp_tolerance** () const
  *<<**extension**>> (p. 116) Getter (see the setter with the same name)*

Static Public Member Functions

- static **DestinationOrder SourceTimestamp** ()

  *Creates a **DestinationOrder** (p. 1258) with DestinationOrderKind::BY_SOURCE_TIMESTAMP.*
- static **DestinationOrder ReceptionTimestamp** ()

  *Creates a **DestinationOrder** (p. 1258) with DestinationOrderKind::BY_RECEPTION_TIMESTAMP.*

8.78.1 Detailed Description

Controls the logical order of updates to the same instance by a **dds::pub::Publisher** (p. 1993).

Entity:

dds::topic::Topic (p. 2474), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Status:

dds::core::status::StatusMask::offered_incompatible_qos() (p. 2380), **dds::core::status::StatusMask↩ ::requested_incompatible_qos()** (p. 2381)

Properties:

RxO (p. 514) = YES
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

8.78.2 Usage

When multiple DataWriters send data for the same topic, the order in which data from different DataWriters are received by the applications of different DataReaders may be different. So different DataReaders may not receive the same "last" value when DataWriters stop sending data.

This QoS policy controls how each subscriber resolves the final value of a data instance that is written by multiple **dds↩ ::pub::DataWriter** (p. 1138) entities (which may be associated with different **dds::pub::Publisher** (p. 1993) entities) running on different nodes.

This QoS can be used to create systems that have the property of "eventual consistency." Thus intermediate states across multiple applications may be inconsistent, but when DataWriters stop sending changes to the same topic, all applications will end up having the same state.

This QoS policy can be set for both DataWriters and DataReaders.

For the DataReader:

The default setting, **dds::core::policy::DestinationOrderKind_def::BY_RECEPTION_TIMESTAMP** (p. 1264), indicates that (assuming the **OWNERSHIP_STRENGTH** (p. 539) policy allows it) the latest received value for the instance should be the one whose value is kept. That is, data will be delivered by a **dds::sub::DataReader** (p. 976) in the order in which it was *received* (which may lead to inconsistent final values).

For **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264), if the scope is set to **rti::core::policy::DestinationOrderScopeKind_def::INSTANCE** (p. 1265) (default), within each instance, the sample's source timestamp shall be used to determine the most recent information. This is the only setting that, in the case of concurrent same-strength DataWriters updating the same instance, ensures that all DataReaders end up with the same final value for the instance. If a DataReader receives a sample for an instance with a source timestamp that is older than the last source timestamp received for the instance, the sample is dropped. The SAMPLE_REJECTED status or the SAMPLE_LOST status will not be updated.

If scope is set to **rti::core::policy::DestinationOrderScopeKind_def::TOPIC** (p. 1265), the ordering is enforced per topic across all instances.

In addition, a DataReader will accept a sample only if the source timestamp is no farther in the future from the reception timestamp than source_timestamp_tolerance. Otherwise, the DDS sample is dropped. The SAMPLE_REJECTED status or the SAMPLE_LOST status will not be updated.

For the DataWriter:

For the default setting, **dds::core::policy::DestinationOrderKind_def::BY_RECEPTION_TIMESTAMP** (p. 1264), the DataWriter will not enforce source timestamp ordering when writing samples using the **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179) or **dds::pub::DataWriter::write(const T&,const dds::core::Time&)** (p. 1149) API. The source timestamp of a new sample can be older than the source timestamp of the previous samples.

When using **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264), If scope is set to **rti::core::policy::DestinationOrderScopeKind_def::INSTANCE** (p. 1265) (default), when writing a sample, the sample's timestamp must not be older than the timestamp of the previously written DDS sample for the same instance. If, however, the timestamp is older than the timestamp of the previously written DDS sample—but the difference is less than the source_timestamp_tolerance—the DDS sample will use the previously written DDS sample's timestamp as its timestamp. Otherwise, if the difference is greater than the tolerance, the write will fail with retcode **dds::core::InvalidArgumentError** (p. 1619).

If scope is set to **rti::core::policy::DestinationOrderScopeKind_def::TOPIC** (p. 1265), a new sample timestamp must not be older than the timestamp of the previously written DDS sample, across all instances. (The ordering is enforced across all instances.)

8.78.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered kind* $>=$ *requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of **dds::core::policy::DestinationOrder::kind** (p. 1261) are considered ordered such that **dds::core::policy::DestinationOrderKind_def::BY_RECEPTION_TIMESTAMP** (p. 1264) $<$ **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264)

8.78.4 Constructor & Destructor Documentation

8.78.4.1 DestinationOrder() [1/2]

```
dds::core::policy::DestinationOrder::DestinationOrder ( )  [inline]
```

Creates the default policy.

8.78.4.2 DestinationOrder() [2/2]

```
dds::core::policy::DestinationOrder::DestinationOrder (
            dds::core::policy::DestinationOrderKind the_kind )  [inline], [explicit]
```

Creates a policy with the specified destination order kind.

8.78.5 Member Function Documentation

8.78.5.1 kind() [1/2]

```
DestinationOrder & dds::core::policy::DestinationOrder::kind (
            dds::core::policy::DestinationOrderKind the_kind )  [inline]
```

Sets the destination order kind.

[default] dds::core::policy::DestinationOrderKind_def::BY_RECEPTION_TIMESTAMP (p. 1264),

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Applied Per-Instance** (p. 287), **Sample Ordering** (p. 234)

8.78.5.2 kind() [2/2]

```
dds::core::policy::DestinationOrderKind dds::core::policy::DestinationOrder::kind ( ) const  [inline]
```

Getter (see the setter with the same name)

8.78.5.3 SourceTimestamp()

```
static DestinationOrder dds::core::policy::DestinationOrder::SourceTimestamp ( ) [inline], [static]
```

Creates a **DestinationOrder** (p. 1258) with DestinationOrderKind::BY_SOURCE_TIMESTAMP.

8.78.5.4 ReceptionTimestamp()

```
static DestinationOrder dds::core::policy::DestinationOrder::ReceptionTimestamp ( ) [inline],
[static]
```

Creates a **DestinationOrder** (p. 1258) with DestinationOrderKind::BY_RECEPTION_TIMESTAMP.

References **dds::core::Duration::from_millisecs()**.

8.78.5.5 scope() [1/2]

```
dds::core::policy::DestinationOrder & scope (
            rti::core::policy::DestinationOrderScopeKind the_scope )
```

<<***extension***>> *(*p. *116)* Sets the destination order scope

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Indicates if tolerance check and the current sample's timestamp is computed based on instance or topic basis.
[default] rti::core::policy::DestinationOrderScopeKind_def::INSTANCE (p. 1265)

Categories

Immutable (p. 243), **Applied Per-Instance** (p. 287), **Sample Ordering** (p. 234)

8.78.5.6 scope() [2/2]

```
rti::core::policy::DestinationOrderScopeKind scope ( ) const
```

<<***extension***>> *(*p. *116)* Getter (see the setter with the same name)

8.78.5.7 source_timestamp_tolerance() [1/2]

```
dds::core::policy::DestinationOrder & source_timestamp_tolerance (
            const  dds::core::Duration & ms )
```

<<***extension***>> *(*p. *116)* Sets the allowed tolerance between source timestamps of consecutive samples.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

When a **dds::pub::DataWriter** (p. 1138) sets **dds::core::policy::DestinationOrderKind_def** (p. 1263) to **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264), when writing a sample, its timestamp must not be less than the timestamp of the previously written sample. However, if it is less than the timestamp of the previously written sample but the difference is less than this tolerance, the sample will use the previously written sample's timestamp as its timestamp. Otherwise, if the difference is greater than this tolerance, the write will fail.

When a **dds::sub::DataReader** (p. 976) sets **dds::core::policy::DestinationOrderKind_def** (p. 1263) to **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264), the **dds::sub::DataReader** (p. 976) will accept a sample only if the source timestamp is no farther in the future from the reception timestamp than this tolerance. Otherwise, the sample is dropped.

[default] 100 milliseconds for **dds::pub::DataWriter** (p. 1138), 30 seconds for **dds::sub::DataReader** (p. 976) and when default-constructed

Categories

Immutable (p. 243), **Applied Per-Instance** (p. 287), **Sample Ordering** (p. 234)

8.78.5.8 source_timestamp_tolerance() [2/2]

```
dds::core::Duration source_timestamp_tolerance ( ) const
```

<<***extension***>> *(*p. *116)* Getter (see the setter with the same name)

8.79 dds::core::policy::DestinationOrderKind_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) DestinationOrderKind.

```
#include <PolicyKind.hpp>
```

Public Types

- enum **type** {
  BY_RECEPTION_TIMESTAMP ,
  BY_SOURCE_TIMESTAMP }

  The underlying `enum` *type.*

8.79.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) DestinationOrderKind.

8.79.2 Member Enumeration Documentation

8.79.2.1 type

`enum` **dds::core::policy::DestinationOrderKind_def::type**

The underlying `enum` type.

Enumerator

BY_RECEPTION_TIMESTAMP	**[default]** Indicates that data is ordered based on the reception time at each **dds::sub::Subscriber** (p. 2410). Since each subscriber may receive the data at different times there is no guarantee that the changes will be seen in the same order. Consequently, it is possible for each subscriber to end up with a different final value for the data.
BY_SOURCE_TIMESTAMP	Indicates that data is ordered based on a time-stamp placed at the source (by RTI Connext or by the application). In any case this guarantees a consistent final value for the data in all subscribers. Note: If Batching is needed along with **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264) and **rti::core::policy::DestinationOrderScopeKind_def::INSTANCE** (p. 1265), then the **rti::core::policy::Batch::source_timestamp_resolution** (p. 885) and **rti::core::policy::Batch::thread_safe_write** (p. 886) setting of **rti::core::policy::Batch** (p. 880) should be set to **dds::core::Duration::zero()** (p. 1450) and true respectively.

8.80 rti::core::policy::DestinationOrderScopeKind_def Struct Reference

<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) DestinationOrderScopeKind

`#include <PolicyKind.hpp>`

Public Types

- enum **type** {
  INSTANCE ,
  TOPIC }

  The underlying `enum` type.

8.80.1 Detailed Description

<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) DestinationOrderScopeKind

8.80.2 Member Enumeration Documentation

8.80.2.1 type

`enum` **`rti::core::policy::DestinationOrderScopeKind_def::type`**

The underlying `enum` type.

Enumerator

INSTANCE	**[default]** Indicates that data is ordered on a per instance basis if used along with **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264). The source timestamp of the current sample is compared to the source timestamp of the previously received sample for the same instance. The tolerance check is also applied per instance.
TOPIC	Indicates that data is ordered on a per topic basis if used along with **dds::core::policy::DestinationOrderKind_def::BY_SOURCE_TIMESTAMP** (p. 1264). The source timestamp of the current sample is compared to the source timestamp of the previously received sample for the same topic. The tolerance check is also applied per topic.

8.81 rti::test::DisableMonitoringLibraryGuard Struct Reference

RAII scope guard to temporarily disable monitoring library.

`#include <utilities.hpp>`

8.81.1 Detailed Description

RAII scope guard to temporarily disable monitoring library.

This class disables monitoring on the DomainParticipantFactory for the duration of its lifetime and restores the original settings when destroyed.

8.82 rti::core::policy::Discovery Class Reference

<<***extension***>> *(p. 116)* Configures entity discovery

`#include <rti/core/policy/CorePolicy.hpp>`

Public Member Functions

- **Discovery** ()

  Creates the default policy.
- **Discovery** & **enabled_transports** (const **dds::core::StringSeq** &the_enabled_transports)

  Sets the transports (by their aliases) available for the discovery mechanism.
- **dds::core::StringSeq enabled_transports** () const

  Getter (see setter with the same name)
- **Discovery** & **initial_peers** (const **dds::core::StringSeq** &the_initial_peers)

  Sets the initial list of peers that the discovery mechanism will contact to announce this DomainParticipant.
- **dds::core::StringSeq initial_peers** () const

  Getter (see setter with the same name)
- **Discovery** & **multicast_receive_addresses** (const **dds::core::StringSeq** &the_multicast_receive_addresses)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the DomainParticipant.*
- **dds::core::StringSeq multicast_receive_addresses** () const

  Getter (see setter with the same name)
- **Discovery** & **metatraffic_transport_priority** (int32_t the_metatraffic_transport_priority)

  *<<**extension**>> (p. 116) The transport priority to use for the **Discovery** (p. 1265) meta-traffic.*
- int32_t **metatraffic_transport_priority** () const

  Getter (see setter with the same name)
- **Discovery** & **accept_unknown_peers** (bool the_accept_unknown_peers)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) Whether to accept a new participant that is not in the initial peers list.*
- bool **accept_unknown_peers** () const

  Getter (see setter with the same name)
- **Discovery** & **enable_endpoint_discovery** (bool the_enable_endpoint_discovery)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Whether to automatically enable endpoint discovery for all the remote participants.*
- bool **enable_endpoint_discovery** () const

  Getter (see setter with the same name)

8.82.1 Detailed Description

*<<**extension**>> (p. 116)* Configures entity discovery

Entity:

dds::domain::DomainParticipant (p. 1326)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.82.2 Usage

This QoS policy identifies where on the network this application can *potentially* discover other applications with which to communicate.

The middleware will periodically send network packets to these locations, announcing itself to any remote applications that may be present, and will listen for announcements from those applications.

This QoS policy is an extension to the DDS standard.

See also

NDDS_DISCOVERY_PEERS (p. 559)

rti::core::policy::DiscoveryConfig (p. 1271)

8.82.3 Constructor & Destructor Documentation

8.82.3.1 Discovery()

```
rti::core::policy::Discovery::Discovery ( )
```

Creates the default policy.

8.82.4 Member Function Documentation

8.82.4.1 enabled_transports() [1/2]

```
Discovery & rti::core::policy::Discovery::enabled_transports (
            const  dds::core::StringSeq & the_enabled_transports )
```

Sets the transports (by their aliases) available for the discovery mechanism.

Only these transports can be used by the discovery mechanism to send meta-traffic via the builtin endpoints (built-in **dds::sub::DataReader** (p. 976) and **dds::pub::DataWriter** (p. 1138)).

Also determines the unicast addresses on which the **Discovery** (p. 1265) mechanism will listen for meta-traffic. These along with the `domain_id` and `participant_id` determine the unicast locators on which the **Discovery** (p. 1265) mechanism can receive meta-data.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 548). These alias names are case sensitive and should be written in lowercase.

[default] Empty sequence. All the transports available to the DomainParticipant are available for use by the **Discovery** (p. 1265) mechanism.

[range] Sequence of non-null,non-empty strings.

Categories

Immutable (p. 243), **Transports** (p. 237), **Discovery** (p. 177)

8.82.4.2 enabled_transports() [2/2]

```
dds::core::StringSeq rti::core::policy::Discovery::enabled_transports ( ) const
```

Getter (see setter with the same name)

8.82.4.3 initial_peers() [1/2]

```
Discovery & rti::core::policy::Discovery::initial_peers (
            const  dds::core::StringSeq & the_initial_peers )
```

Sets the initial list of peers that the discovery mechanism will contact to announce this DomainParticipant.

As part of the participant discovery phase, the **dds::domain::DomainParticipant** (p. 1326) will announce itself to the domain by sending participant DATA messages. The initial_peers specifies the initial list of peers that will be contacted. A remote **dds::domain::DomainParticipant** (p. 1326) is discovered by receiving participant announcements from a remote peer. When the new remote **dds::domain::DomainParticipant** (p. 1326) has been added to the participant's database, the endpoint discovery phase commences and information about the DataWriters and DataReaders is exchanged.

Each element of this list must be a peer descriptor in the proper format (see **Peer Descriptor Format** (p. 560)).

[default] builtin.udpv4://239.255.0.1, builtin.udpv4://127.0.0.1, builtin.shmem:// (See also **NDDS_DISCOVERY_PEERS** (p. 559))

[range] Sequence of arbitrary length.

See also

Peer Descriptor Format (p. 560)

dds::domain::DomainParticipant::add_peer(const std::string & peer_descr_string) (p. 1352)()

Categories

Immutable (p. 243), **Transports** (p. 237), **Discovery Isolation** (p. 183), **Discovery** (p. 177)

8.82.4.4 initial_peers() [2/2]

```
dds::core::StringSeq rti::core::policy::Discovery::initial_peers ( ) const
```

Getter (see setter with the same name)

8.82.4.5 multicast_receive_addresses() [1/2]

```
Discovery & rti::core::policy::Discovery::multicast_receive_addresses (
            const  dds::core::StringSeq & the_multicast_receive_addresses )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the DomainParticipant.

Each element of this list must be a valid multicast address (IPv4 or IPv6) in the proper format (see **Address Format** (p. 561)).

The `domain_id` determines the multicast port on which the **Discovery** (p. 1265) mechanism can receive meta-data.

If NDDS_DISCOVERY_PEERS does *not* contain a multicast address, then the string sequence **rti::core::policy::Discovery::multicast_receive_addresses** (p. 1268) is cleared and the RTI discovery process will not listen for discovery messages via multicast.

If NDDS_DISCOVERY_PEERS contains one or more multicast addresses, the addresses will be stored in **rti::core::policy::Discovery::multicast_receive_addresses** (p. 1268), starting at element 0. They will be stored in the order they appear NDDS_DISCOVERY_PEERS.

Note: Currently, RTI Connext will only listen for discovery traffic on the first multicast address (element 0) in **rti::core::policy::Discovery::multicast_receive_addresses** (p. 1268).

[default] builtin.udpv4://239.255.0.1 (See also **NDDS_DISCOVERY_PEERS** (p. 559))

[range] Sequence of length [0,1], whose elements are multicast addresses. Currently only the first multicast address (if any) is used. The rest are ignored.

See also

Address Format (p. 561)

8.82.4.6 multicast_receive_addresses() [2/2]

```
dds::core::StringSeq rti::core::policy::Discovery::multicast_receive_addresses ( ) const
```

Getter (see setter with the same name)

8.82.4.7 metatraffic_transport_priority() [1/2]

```
Discovery & rti::core::policy::Discovery::metatraffic_transport_priority (
            int32_t the_metatraffic_transport_priority )
```

<<***extension***>> *(*p. *116)* The transport priority to use for the **Discovery** (p. 1265) meta-traffic.

The discovery metatraffic will be sent by the built-in **dds::pub::DataWriter** (p. 1138) using this transport priority.

[default] 0

[range] [0, MAX_UINT]

Categories

Immutable (p. 243), **Transports** (p. 237), **Concurrency** (p. 164), **Discovery** (p. 177)

8.82.4.8 metatraffic_transport_priority() [2/2]

```
int32_t rti::core::policy::Discovery::metatraffic_transport_priority ( ) const
```

Getter (see setter with the same name)

8.82.4.9 accept_unknown_peers() [1/2]

```
Discovery & rti::core::policy::Discovery::accept_unknown_peers (
            bool the_accept_unknown_peers )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Whether to accept a new participant that is not in the initial peers list.

If false, the participant will only communicate with those in the initial peers list and those added via **dds::domain::DomainParticipant::add_peer(const std::string & peer_descr_string)** (p. 1352)().

If true, the participant will also communicate with all discovered remote participants.

Note: If accept_unknown_peers is false and shared memory is disabled, applications on the same node will *not* communicate if only 'localhost' is specified in the peers list. If shared memory is disabled or 'shmem://' is not specified in the peers list, to communicate with other applications on the same node through the loopback interface, you must put the actual node address or hostname in **NDDS_DISCOVERY_PEERS** (p. 559).

[default] true

Categories

Immutable (p. 243), **Transports** (p. 237), **Discovery Isolation** (p. 183), **Discovery** (p. 177)

8.82.4.10 accept_unknown_peers() [2/2]

```
bool rti::core::policy::Discovery::accept_unknown_peers ( ) const
```

Getter (see setter with the same name)

8.82.4.11 enable_endpoint_discovery() [1/2]

```
Discovery & rti::core::policy::Discovery::enable_endpoint_discovery (
            bool the_enable_endpoint_discovery )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Whether to automatically enable endpoint discovery for all the remote participants.

If true, endpoint discovery will automatically occur for every discovered remote participant.

If false, endpoint discovery will be initially disabled and manual activation is required for each discovered participant by calling **dds::domain::DomainParticipant::resume_endpoint_discovery** (p. 1356).

[default] true

Categories

Immutable (p. 243), **Transports** (p. 237), **Discovery Isolation** (p. 183), **Discovery** (p. 177)

8.82.4.12 enable_endpoint_discovery() [2/2]

```
bool rti::core::policy::Discovery::enable_endpoint_discovery ( ) const
```

Getter (see setter with the same name)

8.83 rti::core::policy::DiscoveryConfig Class Reference

<<***extension***>> *(p. 116)* Configures the discovery mechanism

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **DiscoveryConfig** ()

  Creates the default policy.
- **DiscoveryConfig** & **participant_liveliness_lease_duration** (const **dds::core::Duration** &the_participant_↩ liveliness_lease_duration)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) The liveliness lease duration for the participant.*
- **dds::core::Duration participant_liveliness_lease_duration** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **participant_liveliness_assert_period** (const **dds::core::Duration** &the_participant_↩ liveliness_assert_period)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) The period to assert liveliness for the participant.*
- **dds::core::Duration participant_liveliness_assert_period** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **participant_announcement_period** (const **dds::core::Duration** &the_participant_↩ announcement_period)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) The period at which a participant announces itself to potential peers when using the Simple Participant **Discovery** (p. 1265) Protocol 2.0 (SPDP2).*
- **dds::core::Duration participant_announcement_period** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **remote_participant_purge_kind** (**RemoteParticipantPurgeKind** the_remote_↩ participant_purge_kind)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.*
- **RemoteParticipantPurgeKind remote_participant_purge_kind** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **max_liveliness_loss_detection_period** (const **dds::core::Duration** &the_max_↩ liveliness_loss_detection_period)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.*
- **dds::core::Duration max_liveliness_loss_detection_period** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **initial_participant_announcements** (int32_t the_initial_participant_announcements)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) The number of initial announcements sent when a participant is first enabled.*
- int32_t **initial_participant_announcements** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **new_remote_participant_announcements** (int32_t the_new_remote_participant_↩ announcements)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) The number of participant announcements sent when a remote participant is newly discovered.*
- int32_t **new_remote_participant_announcements** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **min_initial_participant_announcement_period** (const **dds::core::Duration** &the_min↩ _initial_participant_announcement_period)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.*
- **dds::core::Duration min_initial_participant_announcement_period** () const

  Getter (see setter with the same name)

- **DiscoveryConfig** & **max_initial_participant_announcement_period** (const **dds::core::Duration** &the_↩ max_initial_participant_announcement_period)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.*
- **dds::core::Duration max_initial_participant_announcement_period** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **participant_reader_resource_limits** (const **BuiltinTopicReaderResourceLimits** &the↩ _participant_reader_resource_limits)

  *<<**extension**>> (p. 116) Resource limits.*
- const **BuiltinTopicReaderResourceLimits** & **participant_reader_resource_limits** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **participant_reader_resource_limits** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **publication_reader** (const **RtpsReliableReaderProtocol** &the_publication_reader)

  *<<**extension**>> (p. 116) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.*
- const **RtpsReliableReaderProtocol** & **publication_reader** () const

  Getter (see setter with the same name)
- **RtpsReliableReaderProtocol** & **publication_reader** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **publication_reader_resource_limits** (const **BuiltinTopicReaderResourceLimits** &the↩ _publication_reader_resource_limits)

  *<<**extension**>> (p. 116) Resource limits.*
- const **BuiltinTopicReaderResourceLimits** & **publication_reader_resource_limits** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **publication_reader_resource_limits** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **subscription_reader** (const **RtpsReliableReaderProtocol** &the_subscription_reader)

  *<<**extension**>> (p. 116) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.*
- const **RtpsReliableReaderProtocol** & **subscription_reader** () const

  Getter (see setter with the same name)
- **RtpsReliableReaderProtocol** & **subscription_reader** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **subscription_reader_resource_limits** (const **BuiltinTopicReaderResourceLimits** &the_subscription_reader_resource_limits)

  *<<**extension**>> (p. 116) Resource limits.*
- const **BuiltinTopicReaderResourceLimits** & **subscription_reader_resource_limits** () const

  Getter (see setter with the same name)
- **BuiltinTopicReaderResourceLimits** & **subscription_reader_resource_limits** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **publication_writer** (const **RtpsReliableWriterProtocol** &the_publication_writer)

  *<<**extension**>> (p. 116) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.*
- const **RtpsReliableWriterProtocol** & **publication_writer** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **publication_writer** ()

  Getter (see setter with the same name)

- **DiscoveryConfig** & **publication_writer_data_lifecycle** (const **dds::core::policy::WriterDataLifecycle** &the_publication_writer_data_lifecycle)

  *<<**extension**>> (p. 116) Writer data lifecycle settings for a built-in publication writer.*
- const **dds::core::policy::WriterDataLifecycle** & **publication_writer_data_lifecycle** () const

  Getter (see setter with the same name)
- **dds::core::policy::WriterDataLifecycle** & **publication_writer_data_lifecycle** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **subscription_writer** (const **RtpsReliableWriterProtocol** &the_subscription_writer)

  *<<**extension**>> (p. 116) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.*
- const **RtpsReliableWriterProtocol** & **subscription_writer** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **subscription_writer** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **subscription_writer_data_lifecycle** (const **dds::core::policy::WriterDataLifecycle** &the_subscription_writer_data_lifecycle)

  *<<**extension**>> (p. 116) Writer data lifecycle settings for a built-in subscription writer.*
- const **dds::core::policy::WriterDataLifecycle** & **subscription_writer_data_lifecycle** () const

  Getter (see setter with the same name)
- **dds::core::policy::WriterDataLifecycle** & **subscription_writer_data_lifecycle** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **builtin_discovery_plugins** (const **DiscoveryConfigBuiltinPluginKindMask** &the_↩ builtin_discovery_plugins)

  *<<**extension**>> (p. 116) Mask of built-in discovery plugin kinds.*
- **DiscoveryConfigBuiltinPluginKindMask** **builtin_discovery_plugins** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **enabled_builtin_channels** (const **DiscoveryConfigBuiltinChannelKindMask** &the_↩ builtin_discovery_plugins)

  *<<**extension**>> (p. 116) The mask specifying which built-in channels should be enabled.*
- **DiscoveryConfigBuiltinChannelKindMask** **enabled_builtin_channels** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **participant_message_reader_reliability_kind** (**dds::core::policy::ReliabilityKind** the↩ _participant_message_reader_reliability_kind)

  ***[DEPRECATED] (**p. **296)** <<**extension**>> (p. 116) Reliability policy for a built-in participant message reader.*
- **dds::core::policy::ReliabilityKind** **participant_message_reader_reliability_kind** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **participant_message_reader** (const **RtpsReliableReaderProtocol** &the_participant_↩ message_reader)

  *<<**extension**>> (p. 116) RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **rti::core::policy::DiscoveryConfig::participant_message_reader_**↩ **reliability_kind** (p. 1295) is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157).*
- const **RtpsReliableReaderProtocol** & **participant_message_reader** () const

  Getter (see setter with the same name)
- **RtpsReliableReaderProtocol** & **participant_message_reader** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **participant_message_writer** (const **RtpsReliableWriterProtocol** &the_participant_↩ message_writer)

*<<**extension**>> (p. 116) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core::policy::ReliabilityKind_def** (p. 2156).*

- const **RtpsReliableWriterProtocol** & **participant_message_writer** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **participant_message_writer** ()

  Getter (see setter with the same name)

- **DiscoveryConfig** & **publication_writer_publish_mode** (const **PublishMode** &the_publication_writer_↩ publish_mode)

  *<<**extension**>> (p. 116) Publish mode policy for the built-in publication writer.*

- const **PublishMode** & **publication_writer_publish_mode** () const

  Getter (see setter with the same name)

- **PublishMode** & **publication_writer_publish_mode** ()

  Getter (see setter with the same name)

- **DiscoveryConfig** & **subscription_writer_publish_mode** (const **PublishMode** &the_subscription_writer_↩ publish_mode)

  *<<**extension**>> (p. 116) Publish mode policy for the built-in subscription writer.*

- const **PublishMode** & **subscription_writer_publish_mode** () const

  Getter (see setter with the same name)

- **PublishMode** & **subscription_writer_publish_mode** ()

  Getter (see setter with the same name)

- **DiscoveryConfig** & **asynchronous_publisher** (const **AsynchronousPublisher** &the_asynchronous_↩ publisher)

  *<<**extension**>> (p. 116) Asynchronous publishing settings for the discovery **dds::pub::Publisher** (p. 1993) and all entities that are created by it.*

- const **AsynchronousPublisher** & **asynchronous_publisher** () const

  Getter (see setter with the same name)

- **AsynchronousPublisher** & **asynchronous_publisher** ()

  Getter (see setter with the same name)

- **DiscoveryConfig** & **default_domain_announcement_period** (const **dds::core::Duration** &the_default_↩ domain_announcement_period)

  *<<**extension**>> (p. 116) The period to announce a participant to the default domain 0.*

- **dds::core::Duration** **default_domain_announcement_period** () const

  Getter (see setter with the same name)

- **DiscoveryConfig** & **ignore_default_domain_announcements** (bool the_ignore_default_domain_↩ announcements)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.*

- bool **ignore_default_domain_announcements** () const

  Getter (see setter with the same name)

- **DiscoveryConfig** & **service_request_writer** (const **RtpsReliableWriterProtocol** &the_service_request_↩ writer)

  *<<**extension**>> (p. 116) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **rti::topic::ServiceRequest** (p. 2358) writer.*

- const **RtpsReliableWriterProtocol** & **service_request_writer** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **service_request_writer** ()

  Getter (see setter with the same name)

- **DiscoveryConfig** & **service_request_writer_data_lifecycle** (const **dds::core::policy::WriterDataLifecycle** &lifecycle)

  *<<**extension**>> (p. 116) Writer data lifecycle settings for a built-in **rti::topic::ServiceRequest** (p. 2358) writer.*
- const **dds::core::policy::WriterDataLifecycle** & **service_request_writer_data_lifecycle** () const

  Getter (see setter with the same name)
- **dds::core::policy::WriterDataLifecycle** & **service_request_writer_data_lifecycle** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **service_request_writer_publish_mode** (const **PublishMode** &the_service_request_writer_publish_mode)

  *<<**extension**>> (p. 116) Publish mode policy for the built-in service request writer.*
- const **PublishMode** & **service_request_writer_publish_mode** () const

  Getter (see setter with the same name)
- **PublishMode** & **service_request_writer_publish_mode** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **service_request_reader** (const **RtpsReliableReaderProtocol** &the_service_request_reader)

  *<<**extension**>> (p. 116) RTPS reliable reader protocol-related configuration settings for a built-in **rti::topic::ServiceRequest** (p. 2358) reader.*
- const **RtpsReliableReaderProtocol** & **service_request_reader** () const

  Getter (see setter with the same name)
- **RtpsReliableReaderProtocol** & **service_request_reader** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **locator_reachability_assert_period** (const **dds::core::Duration** &the_locator_reachability_assert_period)

  *<<**extension**>> (p. 116) Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.*
- **dds::core::Duration locator_reachability_assert_period** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **locator_reachability_lease_duration** (const **dds::core::Duration** &the_locator_reachability_lease_duration)

  *<<**extension**>> (p. 116) The time period after which other DomainParticipants can consider one of their locators as "unreachable" if they do not receive a REACHABILITY PING from this DomainParticipant.*
- **dds::core::Duration locator_reachability_lease_duration** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **locator_reachability_change_detection_period** (const **dds::core::Duration** &the_locator_reachability_change_detection_period)

  *<<**extension**>> (p. 116) Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.*
- **dds::core::Duration locator_reachability_change_detection_period** () const

  Getter (see setter with the same name)
- **DiscoveryConfig** & **secure_volatile_writer** (const **RtpsReliableWriterProtocol** &the_secure_volatile_writer)

  *<<**extension**>> (p. 116) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.*
- const **RtpsReliableWriterProtocol** & **secure_volatile_writer** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **secure_volatile_writer** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **secure_volatile_writer_publish_mode** (const **PublishMode** &the_secure_volatile_writer_publish_mode)

*<<**extension**>> (p. 116) Publish mode policy for the built-in secure volatile writer.*

- const **PublishMode** & **secure_volatile_writer_publish_mode** () const

  Getter (see setter with the same name)

- **PublishMode** & **secure_volatile_writer_publish_mode** ()

  Getter (see setter with the same name)

- **DiscoveryConfig** & **secure_volatile_reader** (const **RtpsReliableReaderProtocol** &the_secure_volatile_↩ reader)

  *<<**extension**>> (p. 116) RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.*

- const **RtpsReliableReaderProtocol** & **secure_volatile_reader** () const

  Getter (see setter with the same name)

- **RtpsReliableReaderProtocol** & **secure_volatile_reader** ()

  Getter (see setter with the same name)

- **DiscoveryConfig** & **endpoint_type_object_lb_serialization_threshold** (int32_t the_endpoint_type_object↩ _lb_serialization_threshold)

  *<<**extension**>> (p. 116) Controls when/if the TypeObject v1 is compressed when propagated in Simple Endpoint **Discovery** (p. 1265).*

- int32_t **endpoint_type_object_lb_serialization_threshold** () const

  Getter (see setter with the same name)

- **DiscoveryConfig** & **dns_tracker_polling_period** (const **dds::core::Duration** &the_dns_tracker_polling_↩ period)

  *<<**extension**>> (p. 116) Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.*

- **dds::core::Duration dns_tracker_polling_period** () const

  Getter (see setter with the same name)

- **DiscoveryConfig** & **participant_configuration_reader** (const **RtpsReliableReaderProtocol** &the_↩ participant_configuration_reader)

  *<<**extension**>> (p. 116) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.*

- const **RtpsReliableReaderProtocol** & **participant_configuration_reader** () const

  Getter (see setter with the same name)

- **RtpsReliableReaderProtocol** & **participant_configuration_reader** ()

  Getter (see setter with the same name)

- **DiscoveryConfig** & **participant_configuration_reader_resource_limits** (const **BuiltinTopicReader**↩ **ResourceLimits** &the_participant_configuration_reader_resource_limits)

  *<<**extension**>> (p. 116) Resource limits for the built-in topic participant configuration reader.*

- const **BuiltinTopicReaderResourceLimits** & **participant_configuration_reader_resource_limits** () const

  Getter (see setter with the same name)

- **BuiltinTopicReaderResourceLimits** & **participant_configuration_reader_resource_limits** ()

  Getter (see setter with the same name)

- **DiscoveryConfig** & **participant_configuration_writer** (const **RtpsReliableWriterProtocol** &the_participant↩ _configuration_writer)

  *<<**extension**>> (p. 116) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.*

- const **RtpsReliableWriterProtocol** & **participant_configuration_writer** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **participant_configuration_writer** ()

  Getter (see setter with the same name)

- **DiscoveryConfig** & **participant_configuration_writer_data_lifecycle** (const **dds::core::policy::Writer↩ DataLifecycle** &the_participant_configuration_writer_data_lifecycle)

  *<<**extension**>> (p. 116) Writer data lifecycle settings for a built-in participant configuration writer.*
- const **dds::core::policy::WriterDataLifecycle** & **participant_configuration_writer_data_lifecycle** () const

  Getter (see setter with the same name)
- **dds::core::policy::WriterDataLifecycle** & **participant_configuration_writer_data_lifecycle** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **participant_configuration_writer_publish_mode** (const **PublishMode** &the_↩ participant_configuration_writer_publish_mode)

  *<<**extension**>> (p. 116) Publish mode policy for the built-in participant configuration writer.*
- const **PublishMode** & **participant_configuration_writer_publish_mode** () const

  Getter (see setter with the same name)
- **PublishMode** & **participant_configuration_writer_publish_mode** ()

  Getter (see setter with the same name)
- **DiscoveryConfig** & **request_types_filter** (const **rti::core::optional_value**< std::string > &the_request_↩ types_filter)

  *<<**extension**>> (p. 116) Topic name pattern for requesting all types via Type Lookup **Service** (p. 2350).*
- **DiscoveryConfig** & **request_types_filter** (const char *the_request_types_filter)

  Setter accepting `const char *` (see other setter)
- **rti::core::optional_value**< std::string > **request_types_filter** () const

  Getter (see setter with the same name)

8.83.1 Detailed Description

*<<**extension**>> (p. 116)* Configures the discovery mechanism

*<<**extension**>> (p. 116)* This QoS policy controls the amount of delay in discovering entities in the system and the amount of discovery traffic in the network.

The amount of network traffic required by the discovery process can vary widely, based on how your application has chosen to configure the middleware's network addressing (e.g., unicast vs. multicast, multicast TTL, etc.), the size of the system, whether all applications are started at the same time or whether start times are staggered, and other factors. Your application can use this policy to make tradeoffs between discovery completion time and network bandwidth utilization. In addition, you can introduce random back-off periods into the discovery process to decrease the probability of network contention when many applications start simultaneously.

Entity:

dds::domain::DomainParticipant (p. 1326)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.83.2 Constructor & Destructor Documentation

8.83.2.1 DiscoveryConfig()

```
rti::core::policy::DiscoveryConfig::DiscoveryConfig ( )  [inline]
```

Creates the default policy.

8.83.3 Member Function Documentation

8.83.3.1 participant_liveliness_lease_duration() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::participant_liveliness_lease_duration (
            const  dds::core::Duration & the_participant_liveliness_lease_duration )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* The liveliness lease duration for the participant.

This is the same as the expiration time of the DomainParticipant as defined in the RTPS protocol.

If the participant has not refreshed its own liveliness to other participants at least once within this period, it may be considered as stale by other participants in the network.

Should be strictly greater than **rti::core::policy::DiscoveryConfig::participant_liveliness_assert_period** (p. 1279).

[default] 100 seconds

[range] [1 nanosec,1 year], > participant_liveliness_assert_period

Categories

Immutable (p. 243), **Discovery** (p. 177), **Liveliness** (p. 199), **Discovery Metadata** (p. 288)

8.83.3.2 participant_liveliness_lease_duration() [2/2]

```
dds::core::Duration rti::core::policy::DiscoveryConfig::participant_liveliness_lease_duration ( )
const
```

Getter (see setter with the same name)

8.83.3.3 participant_liveliness_assert_period() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::participant_liveliness_assert_period (
            const  dds::core::Duration & the_participant_liveliness_assert_period )
```

<<***basic***>> (p. *104)* <<***extension***>> (p. *116)* The period to assert liveliness for the participant.

The period at which the participant will refresh its liveliness to all the peers.

Should be strictly less than **rti::core::policy::DiscoveryConfig::participant_liveliness_lease_duration** (p. 1279).

[default] 30 seconds

[range] [1 nanosec,1 year], < participant_liveliness_lease_duration

Categories

Immutable (p. 243), **Discovery** (p. 177), **Liveliness** (p. 199)

8.83.3.4 participant_liveliness_assert_period() [2/2]

```
dds::core::Duration rti::core::policy::DiscoveryConfig::participant_liveliness_assert_period ( )
const
```

Getter (see setter with the same name)

8.83.3.5 participant_announcement_period() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::participant_announcement_period (
            const  dds::core::Duration & the_participant_announcement_period )
```

<<***basic***>> (p. *104)* <<***extension***>> (p. *116)* The period at which a participant announces itself to potential peers when using the Simple Participant **Discovery** (p. 1265) Protocol 2.0 (SPDP2).

The **rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers)** (p. 1268) list **dds::domain::DomainParticipant::add_peer(const std::string & peer_descr_string)** (p. 1352) API are used to configure a set of potential peers that a DomainParticipant may discover. The **rti::core::policy::DiscoveryConfig::participant_announcement_period** (p. 1280) configures how frequently a DomainParticipant will announce itself to the subset of the configured potential peers that it has not matched with yet. Once a DomainParticipant matches with a DomainParticipant at one of configured potential peer locators, it will no longer announce itself to that locator at this period unless liveliness is lost.

This QoS policy is only supported when using the Simple Participant **Discovery** (p. 1265) Protocol 2.0 (SPDP2). Setting this value when using the Simple Participant **Discovery** (p. 1265) Protocol (SPDP) or other participant discovery protocols is not supported and will result in an error.

[default] dds::core::Duration::automatic() (p. 1451) (Takes the value of **rti::core::policy::DiscoveryConfig::participant_liveliness_assert_period** (p. 1279))

[range] [1 nanosec,1 year]

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.83.3.6 participant_announcement_period() [2/2]

```
dds::core::Duration rti::core::policy::DiscoveryConfig::participant_announcement_period ( ) const
```

Getter (see setter with the same name)

8.83.3.7 remote_participant_purge_kind() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::remote_participant_purge_kind (
            RemoteParticipantPurgeKind the_remote_participant_purge_kind )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.

Most users will not need to change this value from its default, RemoteParticipantPurgeKind::LIVELINESS_BASED↩ _REMOTE_PARTICIPANT_PURGE. However, RemoteParticipantPurgeKind::NO_REMOTE_PARTICIPANT_PURGE may be a good choice if the following conditions apply:

1. **Discovery** (p. 1265) communication with a remote participant may be lost while data communication remains intact. Such will not typically be the case if discovery takes place over the Simple **Discovery** (p. 1265) Protocol, but may be the case if the RTI Enterprise **Discovery** (p. 1265) **Service** (p. 2350) is used.
2. Extensive and prolonged lack of discovery communication between participants is not expected to be common, either because participant loss itself is expected to be rare, or because participants may be lost sporadically but will typically return again.
3. Maintaining inter-participant liveliness is problematic, perhaps because a participant has no writers with the appropriate **dds::core::policy::LivelinessKind_def** (p. 1654).

[default] RemoteParticipantPurgeKind::LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE

Categories

Immutable (p. 243), **Discovery** (p. 177), **Liveliness** (p. 199)

8.83.3.8 remote_participant_purge_kind() [2/2]

```
RemoteParticipantPurgeKind rti::core::policy::DiscoveryConfig::remote_participant_purge_kind ( )
const
```

Getter (see setter with the same name)

8.83.3.9 max_liveliness_loss_detection_period() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::max_liveliness_loss_detection_period (
            const  dds::core::Duration & the_max_liveliness_loss_detection_period )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.

Notification of the loss of liveliness of a remote entity may come more quickly than this duration, depending on the liveliness contract between the local and remote entities and the capabilities of the discovery mechanism in use. For example, a **dds::sub::DataReader** (p. 976) will learn of the loss of liveliness of a matched **dds::pub::DataWriter** (p. 1138) within the reader's offered liveliness lease duration.

Shortening this duration will increase the responsiveness of entities to communication failures. However, it will also increase the CPU usage of the application, as the liveliness of remote entities will be examined more frequently.

[default] 60 seconds

[range] [1 nanosec, 1 year]

Categories

Immutable (p. 243), **Discovery** (p. 177), **Liveliness** (p. 199)

8.83.3.10 max_liveliness_loss_detection_period() [2/2]

```
dds::core::Duration rti::core::policy::DiscoveryConfig::max_liveliness_loss_detection_period ( )
const
```

Getter (see setter with the same name)

8.83.3.11 initial_participant_announcements() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::initial_participant_announcements (
            int32_t the_initial_participant_announcements )
```

<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The number of initial announcements sent when a participant is first enabled.

[default] 5

[range] [1,1 million]

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.83.3.12 initial_participant_announcements() [2/2]

```
int32_t rti::core::policy::DiscoveryConfig::initial_participant_announcements ( ) const
```

Getter (see setter with the same name)

8.83.3.13 new_remote_participant_announcements() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::new_remote_participant_announcements (
            int32_t the_new_remote_participant_announcements )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* The number of participant announcements sent when a remote participant is newly discovered.

These announcements are only sent to the newly discovered remote participant, they are not also broadcast to the initial_peers list.

[default] 2

[range] [0,1 million]

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.83.3.14 new_remote_participant_announcements() [2/2]

```
int32_t rti::core::policy::DiscoveryConfig::new_remote_participant_announcements ( ) const
```

Getter (see setter with the same name)

8.83.3.15 min_initial_participant_announcement_period() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::min_initial_participant_announcement_period
(
            const  dds::core::Duration & the_min_initial_participant_announcement_period )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.

A random delay between this and **rti::core::policy::DiscoveryConfig::max_initial_participant_announcement_period** (p. 1284) is introduced in between initial announcements when a new remote participant is discovered.

The setting of **rti::core::policy::DiscoveryConfig::min_initial_participant_announcement_period** (p. 1283) must be consistent with **rti::core::policy::DiscoveryConfig::max_initial_participant_announcement_period** (p. 1284). For these two values to be consistent, they must verify that:

rti::core::policy::DiscoveryConfig::min_initial_participant_announcement_period (p. 1283) <= **rti::core::policy::DiscoveryConfig::max_initial_participant_announcement_period** (p. 1284).

[default] 10 milliseconds

[range] [1 nanosec,1 year]

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.83.3.16 min_initial_participant_announcement_period() [2/2]

```
dds::core::Duration rti::core::policy::DiscoveryConfig::min_initial_participant_announcement_
period ( ) const
```

Getter (see setter with the same name)

8.83.3.17 max_initial_participant_announcement_period() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::max_initial_participant_announcement_period
(
            const  dds::core::Duration & the_max_initial_participant_announcement_period )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.

A random delay between **rti::core::policy::DiscoveryConfig::min_initial_participant_announcement_period** (p. 1283) and this is introduced in between initial announcements when a new remote participant is discovered.

The setting of **rti::core::policy::DiscoveryConfig::max_initial_participant_announcement_period** (p. 1284) must be consistent with **rti::core::policy::DiscoveryConfig::min_initial_participant_announcement_period** (p. 1283). For these two values to be consistent, they must verify that:

rti::core::policy::DiscoveryConfig::min_initial_participant_announcement_period (p. 1283) <= **rti::core::policy::DiscoveryConfig::max_initial_participant_announcement_period** (p. 1284).

[default] 1 second

[range] [1 nanosec,1 year]

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.83.3.18 max_initial_participant_announcement_period() [2/2]

```
dds::core::Duration rti::core::policy::DiscoveryConfig::max_initial_participant_announcement_
period ( ) const
```

Getter (see setter with the same name)

8.83.3.19 participant_reader_resource_limits() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::participant_reader_resource_limits (
            const  BuiltinTopicReaderResourceLimits & the_participant_reader_resource_limits )
```

<<***extension***>> *(*p. *116)* Resource limits.

Resource limit of the built-in topic participant reader. For details, see **rti::core::policy::BuiltinTopicReaderResourceLimits** (p. 906).

Categories

Immutable (p. 243), **Discovery** (p. 177), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204)

8.83.3.20 participant_reader_resource_limits() [2/3]

```
const  BuiltinTopicReaderResourceLimits & rti::core::policy::DiscoveryConfig::participant_reader↩
_resource_limits ( ) const
```

Getter (see setter with the same name)

8.83.3.21 participant_reader_resource_limits() [3/3]

```
 BuiltinTopicReaderResourceLimits & rti::core::policy::DiscoveryConfig::participant_reader_↩
resource_limits ( )
```

Getter (see setter with the same name)

8.83.3.22 publication_reader() [1/3]

```
 DiscoveryConfig & rti::core::policy::DiscoveryConfig::publication_reader (
            const  RtpsReliableReaderProtocol & the_publication_reader )
```

<<***extension***>> *(*p. *116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.

For details, refer to the **rti::core::RtpsReliableReaderProtocol** (p. 2221)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222)

8.83.3.23 publication_reader() [2/3]

```
const RtpsReliableReaderProtocol & rti::core::policy::DiscoveryConfig::publication_reader ( )
const
```

Getter (see setter with the same name)

8.83.3.24 publication_reader() [3/3]

```
RtpsReliableReaderProtocol & rti::core::policy::DiscoveryConfig::publication_reader ( )
```

Getter (see setter with the same name)

8.83.3.25 publication_reader_resource_limits() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::publication_reader_resource_limits (
            const BuiltinTopicReaderResourceLimits & the_publication_reader_resource_limits )
```

<<***extension***>> *(*p. *116)* Resource limits.

Resource limit of the built-in topic publication reader. For details, see **rti::core::policy::BuiltinTopicReader↩ ResourceLimits** (p. 906).

Categories

Immutable (p. 243), **Discovery** (p. 177), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204)

8.83.3.26 publication_reader_resource_limits() [2/3]

```
const BuiltinTopicReaderResourceLimits & rti::core::policy::DiscoveryConfig::publication_reader↩
_resource_limits ( ) const
```

Getter (see setter with the same name)

8.83.3.27 publication_reader_resource_limits() [3/3]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::DiscoveryConfig::publication_reader_↩
resource_limits ( )
```

Getter (see setter with the same name)

8.83.3.28 subscription_reader() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::subscription_reader (
            const  RtpsReliableReaderProtocol & the_subscription_reader )
```

<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.

For details, refer to the **rti::core::RtpsReliableReaderProtocol** (p. 2221)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222)

8.83.3.29 subscription_reader() [2/3]

```
const  RtpsReliableReaderProtocol & rti::core::policy::DiscoveryConfig::subscription_reader ( )
const
```

Getter (see setter with the same name)

8.83.3.30 subscription_reader() [3/3]

```
RtpsReliableReaderProtocol & rti::core::policy::DiscoveryConfig::subscription_reader ( )
```

Getter (see setter with the same name)

8.83.3.31 subscription_reader_resource_limits() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::subscription_reader_resource_limits (
            const  BuiltinTopicReaderResourceLimits & the_subscription_reader_resource_limits )
```

<<***extension***>> *(*p. *116)* Resource limits.

Resource limit of the built-in topic subscription reader. For details, see **rti::core::policy::BuiltinTopicReader↩ ResourceLimits** (p. 906).

Categories

Immutable (p. 243), **Discovery** (p. 177), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204)

8.83.3.32 subscription_reader_resource_limits() [2/3]

```
const  BuiltinTopicReaderResourceLimits & rti::core::policy::DiscoveryConfig::subscription_↩
reader_resource_limits ( ) const
```

Getter (see setter with the same name)

8.83.3.33 subscription_reader_resource_limits() [3/3]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::DiscoveryConfig::subscription_reader_↩
resource_limits ( )
```

Getter (see setter with the same name)

8.83.3.34 publication_writer() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::publication_writer (
            const  RtpsReliableWriterProtocol & the_publication_writer )
```

<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.

For details, refer to the **rti::core::policy::RtpsReliableWriterProtocol** (p. 2229)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 3.0 seconds;
fast_heartbeat_period 3.0 seconds;
late_joiner_heartbeat_period 3.0 seconds;
virtual_heartbeat_period **dds::core::Duration::infinite()** (p. 1451);
samples_per_virtual_heartbeat **dds::core::LENGTH_UNLIMITED** (p. 438);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers false;
heartbeats_per_max_samples 8;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 131072 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration true;
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **dds::core::LENGTH_UNLIMITED** (p. 438);
max_send_window_size **dds::core::LENGTH_UNLIMITED** (p. 438);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat false;
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat false;

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222)

8.83.3.35 publication_writer() [2/3]

```
const  RtpsReliableWriterProtocol & rti::core::policy::DiscoveryConfig::publication_writer ( )
const
```

Getter (see setter with the same name)

8.83.3.36 publication_writer() [3/3]

```
RtpsReliableWriterProtocol & rti::core::policy::DiscoveryConfig::publication_writer ( )
```

Getter (see setter with the same name)

8.83.3.37 publication_writer_data_lifecycle() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::publication_writer_data_lifecycle (
            const  dds::core::policy::WriterDataLifecycle & the_publication_writer_data_lifecycle
)
```

<<***extension***>> *(*p. *116)* Writer data lifecycle settings for a built-in publication writer.

For details, refer to the **dds::core::policy::WriterDataLifecycle** (p. 2660). **dds::core::policy::WriterDataLifecycle**↩ **::autodispose_unregistered_instances** (p. 2661) will always be forced to true.

Categories

Immutable (p. 243), **Discovery** (p. 177), **Instance Lifecycle** (p. 193)

8.83.3.38 publication_writer_data_lifecycle() [2/3]

```
const  dds::core::policy::WriterDataLifecycle & rti::core::policy::DiscoveryConfig::publication_↩
writer_data_lifecycle ( ) const
```

Getter (see setter with the same name)

8.83.3.39 publication_writer_data_lifecycle() [3/3]

```
dds::core::policy::WriterDataLifecycle & rti::core::policy::DiscoveryConfig::publication_writer↩
_data_lifecycle ( )
```

Getter (see setter with the same name)

8.83.3.40 subscription_writer() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::subscription_writer (
            const  RtpsReliableWriterProtocol & the_subscription_writer )
```

<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.

For details, refer to the **rti::core::policy::RtpsReliableWriterProtocol** (p. 2229)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 3.0 seconds;
fast_heartbeat_period 3.0 seconds;
late_joiner_heartbeat_period 3.0 seconds;
virtual_heartbeat_period **dds::core::Duration::infinite()** (p. 1451);
samples_per_virtual_heartbeat **dds::core::LENGTH_UNLIMITED** (p. 438);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers false;
heartbeats_per_max_samples 8;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 131072 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration true;
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **dds::core::LENGTH_UNLIMITED** (p. 438);
max_send_window_size **dds::core::LENGTH_UNLIMITED** (p. 438);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat false;
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat false;

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222)

8.83.3.41 subscription_writer() [2/3]

```
const  RtpsReliableWriterProtocol & rti::core::policy::DiscoveryConfig::subscription_writer ( )
const
```

Getter (see setter with the same name)

8.83.3.42 subscription_writer() [3/3]

```
RtpsReliableWriterProtocol & rti::core::policy::DiscoveryConfig::subscription_writer ( )
```

Getter (see setter with the same name)

8.83.3.43 subscription_writer_data_lifecycle() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::subscription_writer_data_lifecycle (
            const  dds::core::policy::WriterDataLifecycle & the_subscription_writer_data_lifecycle
)
```

<<***extension***>> *(p. 116)* Writer data lifecycle settings for a built-in subscription writer.

For details, refer to the **dds::core::policy::WriterDataLifecycle** (p. 2660). **dds::core::policy::WriterDataLifecycle↩ ::autodispose_unregistered_instances** (p. 2661) will always be forced to true.

Categories

Immutable (p. 243), **Discovery** (p. 177), **Instance Lifecycle** (p. 193)

8.83.3.44 subscription_writer_data_lifecycle() [2/3]

```
const  dds::core::policy::WriterDataLifecycle & rti::core::policy::DiscoveryConfig::subscription↩
_writer_data_lifecycle ( ) const
```

Getter (see setter with the same name)

8.83.3.45 subscription_writer_data_lifecycle() [3/3]

```
dds::core::policy::WriterDataLifecycle & rti::core::policy::DiscoveryConfig::subscription_↩
writer_data_lifecycle ( )
```

Getter (see setter with the same name)

8.83.3.46 builtin_discovery_plugins() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::builtin_discovery_plugins (
            const  DiscoveryConfigBuiltinPluginKindMask & the_builtin_discovery_plugins )
```

<<***extension***>> *(p. 116)* Mask of built-in discovery plugin kinds.

There are several built-in discovery plugins. This mask enables the different plugins. Any plugin not enabled will not be created.

[default] rti::core::policy::DiscoveryConfigBuiltinPluginKindMask::SDP (p. 1325)

See also

DiscoveryConfigBuiltinPluginKindMask (p. 1322)

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.83.3.47 builtin_discovery_plugins() [2/2]

```
DiscoveryConfigBuiltinPluginKindMask rti::core::policy::DiscoveryConfig::builtin_discovery_↩
plugins ( ) const
```

Getter (see setter with the same name)

8.83.3.48 enabled_builtin_channels() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::enabled_builtin_channels (
            const  DiscoveryConfigBuiltinChannelKindMask & the_builtin_discovery_plugins )
```

<<***extension***>> *(p. 116)* The mask specifying which built-in channels should be enabled.

While there are a number of built-in channels that are used by Connext DDS, the only built-in channels which can currently be enabled or disabled are the **Service** (p. 2350) Request Channel and the Type Lookup **Service** (p. 2350) Channel.

If **rti::core::policy::DomainParticipantResourceLimits::type_object_max_serialized_length** (p. 1419) is set to 0 the middleware will automatically disable the **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::type_↩ lookup_service** (p. 1321) built-in channel.

[default] **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::service_request** (p. 1321) | **rti::core↩ ::policy::DiscoveryConfigBuiltinChannelKindMask::type_lookup_service** (p. 1321)

Categories

Immutable (p. 243)

8.83.3.49 enabled_builtin_channels() [2/2]

```
DiscoveryConfigBuiltinChannelKindMask rti::core::policy::DiscoveryConfig::enabled_builtin_↩
channels ( ) const
```

Getter (see setter with the same name)

8.83.3.50 participant_message_reader_reliability_kind() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::participant_message_reader_reliability_kind
(
            dds::core::policy::ReliabilityKind the_participant_message_reader_reliability_kind )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Reliability policy for a built-in participant message reader.

For details, refer to the **dds::core::policy::ReliabilityKind_def** (p. 2156).

[default] dds::core::policy::ReliabilityKind_def::BEST_EFFORT (p. 2157)

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222), **Liveliness** (p. 199)

8.83.3.51 participant_message_reader_reliability_kind() [2/2]

```
dds::core::policy::ReliabilityKind rti::core::policy::DiscoveryConfig::participant_message_↩
reader_reliability_kind ( ) const
```

Getter (see setter with the same name)

8.83.3.52 participant_message_reader() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::participant_message_reader (
            const  RtpsReliableReaderProtocol & the_participant_message_reader )
```

<<***extension***>> *(p. 116)* RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **rti::core::policy::DiscoveryConfig::participant_message_reader_↩ reliability_kind** (p. 1295) is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157).

For details, refer to the **rti::core::RtpsReliableReaderProtocol** (p. 2221)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222), **Liveliness** (p. 199)

8.83.3.53 participant_message_reader() [2/3]

```
const  RtpsReliableReaderProtocol & rti::core::policy::DiscoveryConfig::participant_message_reader
( ) const
```

Getter (see setter with the same name)

8.83.3.54 participant_message_reader() [3/3]

```
RtpsReliableReaderProtocol & rti::core::policy::DiscoveryConfig::participant_message_reader ( )
```

Getter (see setter with the same name)

8.83.3.55 participant_message_writer() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::participant_message_writer (
            const  RtpsReliableWriterProtocol & the_participant_message_writer )
```

<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) **dds::core::policy::ReliabilityKind_def** (p. 2156).

For details, refer to the **rti::core::policy::RtpsReliableWriterProtocol** (p. 2229)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 1.0 seconds;
fast_heartbeat_period 1.0 seconds;
late_joiner_heartbeat_period 1.0 seconds;
virtual_heartbeat_period **dds::core::Duration::infinite()** (p. 1451);
samples_per_virtual_heartbeat **dds::core::LENGTH_UNLIMITED** (p. 438);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers false;
heartbeats_per_max_samples 1;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 9216 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration true;
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **dds::core::LENGTH_UNLIMITED** (p. 438);
max_send_window_size **dds::core::LENGTH_UNLIMITED** (p. 438);
send_window_update_period 1s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat false;
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat false;

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222), **Liveliness** (p. 199)

8.83.3.56 participant_message_writer() [2/3]

```
const  RtpsReliableWriterProtocol & rti::core::policy::DiscoveryConfig::participant_message_writer
( ) const
```

Getter (see setter with the same name)

8.83.3.57 participant_message_writer() [3/3]

```
RtpsReliableWriterProtocol & rti::core::policy::DiscoveryConfig::participant_message_writer ( )
```

Getter (see setter with the same name)

8.83.3.58 publication_writer_publish_mode() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::publication_writer_publish_mode (
            const  PublishMode & the_publication_writer_publish_mode )
```

<<***extension***>> *(p. 116)* Publish mode policy for the built-in publication writer.

Determines whether the **Discovery** (p. 1265) built-in publication **dds::pub::DataWriter** (p. 1138) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 243), **Discovery** (p. 177), **Flow Control** (p. 188)

8.83.3.59 publication_writer_publish_mode() [2/3]

```
const  PublishMode & rti::core::policy::DiscoveryConfig::publication_writer_publish_mode ( ) const
```

Getter (see setter with the same name)

8.83.3.60 publication_writer_publish_mode() [3/3]

```
PublishMode & rti::core::policy::DiscoveryConfig::publication_writer_publish_mode ( )
```

Getter (see setter with the same name)

8.83.3.61 subscription_writer_publish_mode() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::subscription_writer_publish_mode (
            const  PublishMode & the_subscription_writer_publish_mode )
```

<<***extension***>> *(p. 116)* Publish mode policy for the built-in subscription writer.

Determines whether the **Discovery** (p. 1265) built-in subscription **dds::pub::DataWriter** (p. 1138) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 243), **Discovery** (p. 177), **Flow Control** (p. 188)

8.83.3.62 subscription_writer_publish_mode() [2/3]

```
const  PublishMode & rti::core::policy::DiscoveryConfig::subscription_writer_publish_mode ( )
const
```

Getter (see setter with the same name)

8.83.3.63 subscription_writer_publish_mode() [3/3]

```
PublishMode & rti::core::policy::DiscoveryConfig::subscription_writer_publish_mode ( )
```

Getter (see setter with the same name)

8.83.3.64 asynchronous_publisher() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::asynchronous_publisher (
            const  AsynchronousPublisher & the_asynchronous_publisher )
```

<<***extension***>> *(p. 116)* Asynchronous publishing settings for the discovery **dds::pub::Publisher** (p. 1993) and all entities that are created by it.

Categories

Immutable (p. 243), **Discovery** (p. 177), **Flow Control** (p. 188)

8.83.3.65 asynchronous_publisher() [2/3]

```
const  AsynchronousPublisher & rti::core::policy::DiscoveryConfig::asynchronous_publisher ( )
const
```

Getter (see setter with the same name)

8.83.3.66 asynchronous_publisher() [3/3]

```
AsynchronousPublisher & rti::core::policy::DiscoveryConfig::asynchronous_publisher ( )
```

Getter (see setter with the same name)

8.83.3.67 default_domain_announcement_period() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::default_domain_announcement_period (
            const  dds::core::Duration & the_default_domain_announcement_period )
```

<<***extension***>> *(p. 116)* The period to announce a participant to the default domain 0.

The period at which a participant will announce itself to the default domain 0 using the default UDPv4 multicast group address for discovery traffic on that domain.

For domain 0, the default discovery multicast address is 239.255.0.1:7400.

To disable announcement to the default domain, set this period to **dds::core::Duration::infinite()** (p. 1451).

When this period is set to a value other than **dds::core::Duration::infinite()** (p. 1451) and **rti::core::policy::↩ DiscoveryConfig::ignore_default_domain_announcements** (p. 1301) is set to false, you can get information about participants running in different domains by creating a participant in domain 0 and implementing the on_data_available callback in the **dds::topic::ParticipantBuiltinTopicData** (p. 1908) built-in DataReader's listener.

You can learn the domain ID associated with a participant by looking at the field **dds::topic::ParticipantBuiltinTopic↩ Data::domain_id** (p. 1914).

[default] 30 seconds

[range] [1 nanosec,1 year] or **dds::core::Duration::infinite()** (p. 1451)

See also

dds::topic::ParticipantBuiltinTopicData::domain_id (p. 1914)

rti::core::policy::DiscoveryConfig::ignore_default_domain_announcements (p. 1301)

Categories

Immutable (p. 243), **Discovery** (p. 177), **Debuggability** (p. 172)

8.83.3.68 default_domain_announcement_period() [2/2]

```
dds::core::Duration rti::core::policy::DiscoveryConfig::default_domain_announcement_period ( )
const
```

Getter (see setter with the same name)

8.83.3.69 ignore_default_domain_announcements() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::ignore_default_domain_announcements (
            bool the_ignore_default_domain_announcements )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.

This setting only applies to participants running on the default domain 0 and using the default port mapping.

When this setting is set to true, a participant running on the default domain 0 will ignore announcements from participants running on different domain IDs.

When this setting is set to false, a participant running on the default domain 0 will provide announcements from participants running on different domain IDs to the application via the **dds::topic::ParticipantBuiltinTopicData** (p. 1908) built-in DataReader.

[default] true

See also

dds::topic::ParticipantBuiltinTopicData::domain_id (p. 1914)

rti::core::policy::DiscoveryConfig::default_domain_announcement_period (p. 1300)

Categories

Immutable (p. 243), **Discovery** (p. 177), **Debuggability** (p. 172)

8.83.3.70 ignore_default_domain_announcements() [2/2]

```
bool rti::core::policy::DiscoveryConfig::ignore_default_domain_announcements ( ) const
```

Getter (see setter with the same name)

8.83.3.71 service_request_writer() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::service_request_writer (
            const  RtpsReliableWriterProtocol & the_service_request_writer )
```

<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **rti::topic::ServiceRequest** (p. 2358) writer.

For details, refer to the **rti::core::policy::RtpsReliableWriterProtocol** (p. 2229)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 3.0 seconds;
fast_heartbeat_period 3.0 seconds;
late_joiner_heartbeat_period 3.0 seconds;
virtual_heartbeat_period **dds::core::Duration::infinite()** (p. 1451);
samples_per_virtual_heartbeat **dds::core::LENGTH_UNLIMITED** (p. 438);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers false;
heartbeats_per_max_samples 8;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 131072 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration true;
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **dds::core::LENGTH_UNLIMITED** (p. 438);
max_send_window_size **dds::core::LENGTH_UNLIMITED** (p. 438);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat false;
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat false;

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222)

8.83.3.72 service_request_writer() [2/3]

```
const  RtpsReliableWriterProtocol & rti::core::policy::DiscoveryConfig::service_request_writer ( )
const
```

Getter (see setter with the same name)

8.83.3.73 service_request_writer() [3/3]

```
RtpsReliableWriterProtocol & rti::core::policy::DiscoveryConfig::service_request_writer ( )
```

Getter (see setter with the same name)

8.83.3.74 service_request_writer_data_lifecycle() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::service_request_writer_data_lifecycle (
            const  dds::core::policy::WriterDataLifecycle & lifecycle )
```

<<***extension***>> *(*p. *116)* Writer data lifecycle settings for a built-in **rti::topic::ServiceRequest** (p. 2358) writer.

For details, refer to the **dds::core::policy::WriterDataLifecycle** (p. 2660).

Categories

Immutable (p. 243), **Discovery** (p. 177), **Instance Lifecycle** (p. 193)

8.83.3.75 service_request_writer_data_lifecycle() [2/3]

```
const  dds::core::policy::WriterDataLifecycle & rti::core::policy::DiscoveryConfig::service_↩
request_writer_data_lifecycle ( ) const
```

Getter (see setter with the same name)

8.83.3.76 service_request_writer_data_lifecycle() [3/3]

```
dds::core::policy::WriterDataLifecycle & rti::core::policy::DiscoveryConfig::service_request_↩
writer_data_lifecycle ( )
```

Getter (see setter with the same name)

8.83.3.77 service_request_writer_publish_mode() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::service_request_writer_publish_mode (
            const  PublishMode & the_service_request_writer_publish_mode )
```

<<***extension***>> *(p. 116)* Publish mode policy for the built-in service request writer.

Determines whether the **Discovery** (p. 1265) built-in service request **dds::pub::DataWriter** (p. 1138) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 243), **Discovery** (p. 177), **Flow Control** (p. 188)

8.83.3.78 service_request_writer_publish_mode() [2/3]

```
const  PublishMode & rti::core::policy::DiscoveryConfig::service_request_writer_publish_mode ( )
const
```

Getter (see setter with the same name)

8.83.3.79 service_request_writer_publish_mode() [3/3]

```
PublishMode & rti::core::policy::DiscoveryConfig::service_request_writer_publish_mode ( )
```

Getter (see setter with the same name)

8.83.3.80 service_request_reader() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::service_request_reader (
            const  RtpsReliableReaderProtocol & the_service_request_reader )
```

<<***extension***>> *(p. 116)* RTPS reliable reader protocol-related configuration settings for a built-in **rti::topic::ServiceRequest** (p. 2358) reader.

For details, refer to the **rti::core::RtpsReliableReaderProtocol** (p. 2221)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222)

8.83.3.81 service_request_reader() [2/3]

```
const  RtpsReliableReaderProtocol & rti::core::policy::DiscoveryConfig::service_request_reader ( )
const
```

Getter (see setter with the same name)

8.83.3.82 service_request_reader() [3/3]

```
RtpsReliableReaderProtocol & rti::core::policy::DiscoveryConfig::service_request_reader ( )
```

Getter (see setter with the same name)

8.83.3.83 locator_reachability_assert_period() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::locator_reachability_assert_period (
            const  dds::core::Duration & the_locator_reachability_assert_period )
```

<<***extension***>> *(p. 116)* Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.

This setting configures the period at which this **dds::domain::DomainParticipant** (p. 1326) will ping all the locators that it has discovered from other DomainParticipants. This period should be strictly less than **rti::core::policy::Discovery↩ Config::locator_reachability_lease_duration** (p. 1306).

If **rti::core::policy::DiscoveryConfig::locator_reachability_lease_duration** (p. 1306) is **dds::core::Duration↩ ::infinite()** (p. 1451) this parameter is ignored. The DomainParticipant will not assert remote locators.

[default] 20 seconds

[range] [1 nanosec,1 year]

See also

rti::core::policy::DiscoveryConfig::locator_reachability_lease_duration (p. 1306)

Categories

Immutable (p. 243), **Transports** (p. 237)

8.83.3.84 locator_reachability_assert_period() [2/2]

```
dds::core::Duration rti::core::policy::DiscoveryConfig::locator_reachability_assert_period ( )
const
```

Getter (see setter with the same name)

8.83.3.85 locator_reachability_lease_duration() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::locator_reachability_lease_duration (
            const  dds::core::Duration & the_locator_reachability_lease_duration )
```

<<***extension***>> *(p. 116)* The time period after which other DomainParticipants can consider one of their locators as "unreachable" if they do not receive a REACHABILITY PING from this DomainParticipant.

For the purpose of this explanation, we will use 'local' to refer to the DomainParticipant in which we configure locator_reachability_lease_duration and 'remote' to refer to the other DomainParticipants communicating with the local DomainParticipant.

This setting configures a timeout announced to the remote DomainParticipants. This timeout is used by the remote DomainParticipants as the maximum period by which a remote locator must be asserted by the local DomainParticipant (through a REACHABILITY PING message) before considering this locator as "unreachable" from the local Domain↩ Participant.

When a remote DomainParticipant detects that one of its locators is not reachable from the local DomainParticipant, it will notify the local DomainParticipant of this event. From that moment on, and until notified otherwise, the local DomainParticipant will not send RTPS messages to remote DomainParticipants using this locator.

If this value is set to **dds::core::Duration::infinite()** (p. 1451), the local DomainParticipant will send RTPS messages to a remote DomainParticipant on the locators announced by the remote DomainParticipant, regardless of whether or not the remote DomainParticipant can be reached using these locators.

[default] dds::core::Duration::infinite() (p. 1451)

[range] [1 nanosec,1 year] or **dds::core::Duration::infinite()** (p. 1451)

8.83.3.86 locator_reachability_lease_duration() [2/2]

```
dds::core::Duration rti::core::policy::DiscoveryConfig::locator_reachability_lease_duration ( )
const
```

Getter (see setter with the same name)

8.83.3.87 locator_reachability_change_detection_period() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::locator_reachability_change_detection_↩
period (
            const  dds::core::Duration & the_locator_reachability_change_detection_period )
```

<<***extension***>> *(p. 116)* Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.

This setting determines the maximum period at which this DomainParticipant will check to see if its locators are reachable from other DomainParticipants according to the other DomainParticipants' **rti::core::policy::DiscoveryConfig↩ ::locator_reachability_lease_duration** (p. 1306) value.

If **rti::core::policy::DiscoveryConfig::locator_reachability_lease_duration** (p. 1306) is **dds::core::Duration↩ ::infinite()** (p. 1451) this parameter is ignored. The DomainParticipant will not schedule an event to see if its locators are reachable from other DomainParticipants.

[default] 60 seconds

[range] [1 nanosec,1 year]

See also

rti::core::policy::DiscoveryConfig::locator_reachability_lease_duration (p. 1306)

Categories

Immutable (p. 243), **Transports** (p. 237)

8.83.3.88 locator_reachability_change_detection_period() [2/2]

```
dds::core::Duration rti::core::policy::DiscoveryConfig::locator_reachability_change_detection_↩
period ( ) const
```

Getter (see setter with the same name)

8.83.3.89 secure_volatile_writer() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::secure_volatile_writer (
            const  RtpsReliableWriterProtocol & the_secure_volatile_writer )
```

<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.

For details, refer to the **rti::core::policy::RtpsReliableWriterProtocol** (p. 2229)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 200.0 millisecond;
fast_heartbeat_period 200.0 milliseconds;
late_joiner_heartbeat_period 200.0 millisecond;
virtual_heartbeat_period **dds::core::Duration::infinite()** (p. 1451);
samples_per_virtual_heartbeat **dds::core::LENGTH_UNLIMITED** (p. 438);
max_heartbeat_retries 1000;
inactivate_nonprogressing_readers false;
heartbeats_per_max_samples 100000000;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 9216 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 millisecond;
disable_positive_acks_max_sample_keep_duration 1.0 second;
disable_positive_acks_enable_adaptive_sample_keep_duration true;
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **dds::core::LENGTH_UNLIMITED** (p. 438);
max_send_window_size **dds::core::LENGTH_UNLIMITED** (p. 438);
send_window_update_period 1.0 second;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat false;
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat false;

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222), **Security** (p. 235)

8.83.3.90 secure_volatile_writer() [2/3]

```
const  RtpsReliableWriterProtocol & rti::core::policy::DiscoveryConfig::secure_volatile_writer ( )
const
```

Getter (see setter with the same name)

8.83.3.91 secure_volatile_writer() [3/3]

```
RtpsReliableWriterProtocol & rti::core::policy::DiscoveryConfig::secure_volatile_writer ( )
```

Getter (see setter with the same name)

8.83.3.92 secure_volatile_writer_publish_mode() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::secure_volatile_writer_publish_mode (
            const  PublishMode & the_secure_volatile_writer_publish_mode )
```

<<***extension***>> *(*p. *116)* Publish mode policy for the built-in secure volatile writer.

Determines whether the built-in secure volatile **dds::pub::DataWriter** (p. 1138) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 243), **Discovery** (p. 177), **Flow Control** (p. 188), **Security** (p. 235)

8.83.3.93 secure_volatile_writer_publish_mode() [2/3]

```
const  PublishMode & rti::core::policy::DiscoveryConfig::secure_volatile_writer_publish_mode ( )
const
```

Getter (see setter with the same name)

8.83.3.94 secure_volatile_writer_publish_mode() [3/3]

```
PublishMode & rti::core::policy::DiscoveryConfig::secure_volatile_writer_publish_mode ( )
```

Getter (see setter with the same name)

8.83.3.95 secure_volatile_reader() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::secure_volatile_reader (
            const  RtpsReliableReaderProtocol & the_secure_volatile_reader )
```

<<***extension***>> *(*p. *116)* RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.

For details, refer to the **rti::core::RtpsReliableReaderProtocol** (p. 2221)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222), **Security** (p. 235)

8.83.3.96 secure_volatile_reader() [2/3]

```
const  RtpsReliableReaderProtocol & rti::core::policy::DiscoveryConfig::secure_volatile_reader ( )
const
```

Getter (see setter with the same name)

8.83.3.97 secure_volatile_reader() [3/3]

```
RtpsReliableReaderProtocol & rti::core::policy::DiscoveryConfig::secure_volatile_reader ( )
```

Getter (see setter with the same name)

8.83.3.98 endpoint_type_object_lb_serialization_threshold() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::endpoint_type_object_lb_serialization_↩
threshold (
            int32_t the_endpoint_type_object_lb_serialization_threshold )
```

<<***extension***>> *(p. 116)* Controls when/if the TypeObject v1 is compressed when propagated in Simple Endpoint **Discovery** (p. 1265).

Minimum size (in bytes) of the serialized TypeObject v1 that will trigger the serialization of a compressed TypeObject v1 (known as TypeObjectLb) instead of the regular TypeObject v1.

For example, setting this property to 1000 will trigger the serialization of the TypeObjectLb for TypeObject v1s whose serialized size is greater than 1000 Bytes.

The sentinel value -1 disables TypeObject v1 compression.

If **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::type_lookup_service** (p. 1321) is also enabled, this threshold does not affect compression of TypeObject v2s in TypeLookup **Service** (p. 2350) messages.

[default] 0. The default value 0 enables TypeObject v1 compression by always sending TypeObjectLb.

[range] [-1, 2147483647]

Categories

Immutable (p. 243), **Discovery** (p. 177), **Sending Large Data** (p. 197), **Type System** (p. 240)

8.83.3.99 endpoint_type_object_lb_serialization_threshold() [2/2]

```
int32_t rti::core::policy::DiscoveryConfig::endpoint_type_object_lb_serialization_threshold ( )
const
```

Getter (see setter with the same name)

8.83.3.100 dns_tracker_polling_period() [1/2]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::dns_tracker_polling_period (
            const  dds::core::Duration & the_dns_tracker_polling_period )
```

<<***extension***>> *(*p. *116)* Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.

RTI Connext allows the use of hostnames instead of IP addresses when configuring initial peers for specific transports (e.g.: UDPv4 and UDPv6). The DNS tracker keeps the IP addresses of these hostnames updated. The DNS tracker builds a list of hostnames from the initial peers of a DomainParticipant, queries the DNS for those hostnames, and updates the resolved IP addresses when the IP addresses change. The frequency of these queries is defined by the DNS tracker polling period.

RTI Connext keeps information regarding the hostnames of peers if they are part of the **rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers)** (p. 1268). The information regarding peers added through the **dds::domain::DomainParticipant::add_peer(const std::string & peer_descr_string)** (p. 1352) operation is kept only if the DNS tracker has been enabled before adding a peer.

The DNS tracker is useful in situations where the IP address of an as-yet undiscovered peer may change. For example, if the **rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers)** (p. 1268) of a participant is set to "hostname1" and the IP address associated with "hostname1" changes after the participant is created, the participant has no way of knowing about that change unless it queries the DNS again.

When two participants have already discovered each other, any IP address change is communicated directly between the participants and there is no need for the DNS tracker. In other words, they do not wait for the DNS tracker polling period to update the IP address–the change is communicated immediately between the participants so that communication can continue uninterrupted.

The value should be (but is not required to be) set to a value that is less than the **rti::core::policy::DiscoveryConfig::participant_liveliness_assert_period** (p. 1279) (if using SPDP) or **rti::core::policy::DiscoveryConfig::participant_liveliness_assert_period** (p. 1279) (is using SPDP2) so that multiple participant announcements are not sent to an outdated IP address.

When the period is set to **dds::core::Duration::infinite()** (p. 1451), the tracker is disabled.

[default] 20 seconds

[range] [1 second,1 year], **dds::core::Duration::infinite()** (p. 1451)

Categories

Immutable (p. 243), **Discovery** (p. 177), **Transports** (p. 237)

8.83.3.101 dns_tracker_polling_period() [2/2]

```
dds::core::Duration rti::core::policy::DiscoveryConfig::dns_tracker_polling_period ( ) const
```

Getter (see setter with the same name)

8.83.3.102 participant_configuration_reader() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::participant_configuration_reader (
            const  RtpsReliableReaderProtocol & the_participant_configuration_reader )
```

<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.

For details, refer to the **rti::core::RtpsReliableReaderProtocol** (p. 2221)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222)

8.83.3.103 participant_configuration_reader() [2/3]

```
const  RtpsReliableReaderProtocol & rti::core::policy::DiscoveryConfig::participant_configuration↩
_reader ( ) const
```

Getter (see setter with the same name)

8.83.3.104 participant_configuration_reader() [3/3]

```
 RtpsReliableReaderProtocol & rti::core::policy::DiscoveryConfig::participant_configuration_reader
( )
```

Getter (see setter with the same name)

8.83.3.105 participant_configuration_reader_resource_limits() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::participant_configuration_reader_resource_
limits (
            const  BuiltinTopicReaderResourceLimits & the_participant_configuration_reader_
resource_limits )
```

<<***extension***>> *(p. 116)* Resource limits for the built-in topic participant configuration reader.

For details, see **rti::core::policy::BuiltinTopicReaderResourceLimits** (p. 906).

Categories

Immutable (p. 243), **Discovery** (p. 177), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204)

8.83.3.106 participant_configuration_reader_resource_limits() [2/3]

```
const  BuiltinTopicReaderResourceLimits & rti::core::policy::DiscoveryConfig::participant_configuration_
reader_resource_limits ( ) const
```

Getter (see setter with the same name)

8.83.3.107 participant_configuration_reader_resource_limits() [3/3]

```
BuiltinTopicReaderResourceLimits & rti::core::policy::DiscoveryConfig::participant_configuration_
reader_resource_limits ( )
```

Getter (see setter with the same name)

8.83.3.108 participant_configuration_writer() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::participant_configuration_writer (
            const  RtpsReliableWriterProtocol & the_participant_configuration_writer )
```

<<***extension***>> *(p. 116)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.

For details, refer to the **rti::core::policy::RtpsReliableWriterProtocol** (p. 2229)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 3.0 seconds;
fast_heartbeat_period 3.0 seconds;
late_joiner_heartbeat_period 3.0 seconds;
virtual_heartbeat_period **dds::core::Duration::infinite()** (p. 1451);
samples_per_virtual_heartbeat **dds::core::LENGTH_UNLIMITED** (p. 438);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers false;
heartbeats_per_max_samples 8;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 131072 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration true;
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **dds::core::LENGTH_UNLIMITED** (p. 438);
max_send_window_size **dds::core::LENGTH_UNLIMITED** (p. 438);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat false;
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat false;

Categories

Immutable (p. 243), **Discovery** (p. 177), **Reliability** (p. 222)

8.83.3.109 participant_configuration_writer() [2/3]

```
const  RtpsReliableWriterProtocol & rti::core::policy::DiscoveryConfig::participant_configuration↩
_writer ( ) const
```

Getter (see setter with the same name)

8.83.3.110 participant_configuration_writer() [3/3]

```
RtpsReliableWriterProtocol & rti::core::policy::DiscoveryConfig::participant_configuration_writer
( )
```

Getter (see setter with the same name)

8.83.3.111 participant_configuration_writer_data_lifecycle() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::participant_configuration_writer_data_↩
lifecycle (
            const  dds::core::policy::WriterDataLifecycle & the_participant_configuration_↩
writer_data_lifecycle )
```

<<***extension***>> *(*p. *116)* Writer data lifecycle settings for a built-in participant configuration writer.

For details, refer to the **dds::core::policy::WriterDataLifecycle** (p. 2660). **dds::core::policy::WriterDataLifecycle↩ ::autodispose_unregistered_instances** (p. 2661) will always be forced to true.

Categories

Immutable (p. 243), **Discovery** (p. 177), **Instance Lifecycle** (p. 193)

8.83.3.112 participant_configuration_writer_data_lifecycle() [2/3]

```
const  dds::core::policy::WriterDataLifecycle & rti::core::policy::DiscoveryConfig::participant_↩
configuration_writer_data_lifecycle ( ) const
```

Getter (see setter with the same name)

8.83.3.113 participant_configuration_writer_data_lifecycle() [3/3]

```
dds::core::policy::WriterDataLifecycle & rti::core::policy::DiscoveryConfig::participant_configuration↩
_writer_data_lifecycle ( )
```

Getter (see setter with the same name)

8.83.3.114 participant_configuration_writer_publish_mode() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::participant_configuration_writer_publish_↩
mode (
            const  PublishMode & the_participant_configuration_writer_publish_mode )
```

<<***extension***>> *(p. 116)* Publish mode policy for the built-in participant configuration writer.

Determines whether the **Discovery** (p. 1265) built-in participant configuration **dds::pub::DataWriter** (p. 1138) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 243), **Discovery** (p. 177), **Flow Control** (p. 188)

8.83.3.115 participant_configuration_writer_publish_mode() [2/3]

```
const  PublishMode & rti::core::policy::DiscoveryConfig::participant_configuration_writer_publish↩
_mode ( ) const
```

Getter (see setter with the same name)

8.83.3.116 participant_configuration_writer_publish_mode() [3/3]

```
PublishMode & rti::core::policy::DiscoveryConfig::participant_configuration_writer_publish_mode (
)
```

Getter (see setter with the same name)

8.83.3.117 request_types_filter() [1/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::request_types_filter (
            const  rti::core::optional_value< std::string > & the_request_types_filter )
```

<<***extension***>> *(p. 116)* Topic name pattern for requesting all types via Type Lookup **Service** (p. 2350).

When using TypeObject v2 (enabled via **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::type_lookup_service** (p. 1321)), this field configures a topic name pattern. Any remote endpoint whose topic name matches this pattern will have all of its types requested by the Type Lookup **Service** (p. 2350), even if no matching local endpoint exists.

By default, the Type Lookup **Service** (p. 2350) only requests unknown types when a matching local endpoint exists. This setting allows you to proactively retrieve type information for remote endpoints matching specific topic names, which can be useful for applications where types are discovered on the wire (such as those using Dynamic Data).

This is a regex string, so you can use regex syntax to match multiple topic names (including wildcards).

If this is NULL or an empty string, the middleware will only request types for remote endpoints that have a matching local endpoint.

If you are using **dds::core::xtypes::DynamicData** (p. 1461) and relying on discovering types on the wire, you should set this to a value that matches the topic names of your remote endpoints to ensure that you receive type information for those endpoints.

[default] NULL

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.83.3.118 request_types_filter() [2/3]

```
DiscoveryConfig & rti::core::policy::DiscoveryConfig::request_types_filter (
            const char * the_request_types_filter )
```

Setter accepting `const char *` (see other setter)

8.83.3.119 request_types_filter() [3/3]

```
rti::core::optional_value< std::string > rti::core::policy::DiscoveryConfig::request_types_filter ( ) const
```

Getter (see setter with the same name)

8.84 rti::core::policy::DiscoveryConfigBuiltinChannelKindMask Class Reference

<<***extension***>> *(p. 354)* A mask that selects the built-in channels to be used

`#include <PolicyKind.hpp>`

Inherits std::bitset< 16 >.

Public Types

- typedef std::bitset< 16 > **MaskType**
  The base type, std::bitset.

Public Member Functions

- **DiscoveryConfigBuiltinChannelKindMask** ()=default
  Creates an empty mask.
- **DiscoveryConfigBuiltinChannelKindMask** (uint64_t mask)
  Creates a mask from the bits in an integer.
- **DiscoveryConfigBuiltinChannelKindMask** (const **MaskType** &mask)
  Creates a mask from a std::bitset.

Static Public Member Functions

- static **DiscoveryConfigBuiltinChannelKindMask** **all** ()
  All the bits are set, indicating that all channels are enabled.
- static **DiscoveryConfigBuiltinChannelKindMask** **none** ()
  No bits are set, indicating that all channels are disabled.
- static **DiscoveryConfigBuiltinChannelKindMask** **service_request** ()
  Set the bits to enable the service request channel (which is the default setting).
- static **DiscoveryConfigBuiltinChannelKindMask** **type_lookup_service** ()
  Set the bits to enable the typelookup service channel.
- static **DiscoveryConfigBuiltinChannelKindMask** **type_lookup_service_request** ()
  Set the bits to enable the typelookup service request channel.
- static **DiscoveryConfigBuiltinChannelKindMask** **type_lookup_service_reply** ()
  Set the bits to enable the typelookup service reply channel.

8.84.1 Detailed Description

<<***extension***>> *(p. 354)* A mask that selects the built-in channels to be used

8.84.2 Member Typedef Documentation

8.84.2.1 MaskType

```
typedef std::bitset<16>  rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::MaskType
```

The base type, std::bitset.

8.84.3 Constructor & Destructor Documentation

8.84.3.1 DiscoveryConfigBuiltinChannelKindMask() [1/3]

```
rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::DiscoveryConfigBuiltinChannelKindMask (
) [default]
```

Creates an empty mask.

Referenced by **all()**, **none()**, **service_request()**, **type_lookup_service()**, **type_lookup_service_reply()**, and **type_lookup_service_request()**.

8.84.3.2 DiscoveryConfigBuiltinChannelKindMask() [2/3]

```
rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::DiscoveryConfigBuiltinChannelKindMask (
            uint64_t mask ) [inline], [explicit]
```

Creates a mask from the bits in an integer.

8.84.3.3 DiscoveryConfigBuiltinChannelKindMask() [3/3]

```
rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::DiscoveryConfigBuiltinChannelKindMask (
            const MaskType & mask ) [inline], [explicit]
```

Creates a mask from a std::bitset.

8.84.4 Member Function Documentation

8.84.4.1 service_request()

```
static DiscoveryConfigBuiltinChannelKindMask rti::core::policy::DiscoveryConfigBuiltinChannel↩
KindMask::service_request ( ) [inline], [static]
```

Set the bits to enable the service request channel (which is the default setting).

Enables the ServiceRequest channel, which is required by the TopicQuery and locator reachability features. Disabling the ServiceRequest channel reduces resource consumption including network bandwidth, CPU utilization, and memory.

References **DiscoveryConfigBuiltinChannelKindMask()**.

8.84.4.2 type_lookup_service()

```
static DiscoveryConfigBuiltinChannelKindMask rti::core::policy::DiscoveryConfigBuiltinChannel↩
KindMask::type_lookup_service ( ) [inline], [static]
```

Set the bits to enable the typelookup service channel.

Enables both the Type Lookup **Service** (p. 2350) Request channel and Type Lookup **Service** (p. 2350) Reply channel.

References **DiscoveryConfigBuiltinChannelKindMask()**.

8.84.4.3 type_lookup_service_request()

```
static DiscoveryConfigBuiltinChannelKindMask rti::core::policy::DiscoveryConfigBuiltinChannel↩
KindMask::type_lookup_service_request ( ) [inline], [static]
```

Set the bits to enable the typelookup service request channel.

Enables the Type Lookup **Service** (p. 2350) Request channel.

References **DiscoveryConfigBuiltinChannelKindMask()**.

8.84.4.4 type_lookup_service_reply()

```
static DiscoveryConfigBuiltinChannelKindMask rti::core::policy::DiscoveryConfigBuiltinChannel↩
KindMask::type_lookup_service_reply ( ) [inline], [static]
```

Set the bits to enable the typelookup service reply channel.

Enables the Type Lookup **Service** (p. 2350) Reply channel.

References **DiscoveryConfigBuiltinChannelKindMask()**.

8.85 rti::core::policy::DiscoveryConfigBuiltinPluginKindMask Class Reference

<<***extension***>> *(p. 354)* A mask that selects the built-in discovery plugins to be used

```
#include <PolicyKind.hpp>
```

Inherits std::bitset< 5 >.

Public Types

- typedef std::bitset< 5 > **MaskType**
  The base type, std::bitset.

Public Member Functions

- **DiscoveryConfigBuiltinPluginKindMask** ()
  Creates an empty mask.
- **DiscoveryConfigBuiltinPluginKindMask** (uint64_t mask)
  Creates a mask from the bits in an integer.
- **DiscoveryConfigBuiltinPluginKindMask** (const **MaskType** &mask)
  Creates a mask from a std::bitset.

Static Public Member Functions

- static const **DiscoveryConfigBuiltinPluginKindMask** **none** ()
  No bits are set.
- static const **DiscoveryConfigBuiltinPluginKindMask** **SPDP** ()
  *Simple Participant **Discovery** (p. 1265) Protocol.*
- static const **DiscoveryConfigBuiltinPluginKindMask** **SEDP** ()
  *Simple Endpoint **Discovery** (p. 1265) Protocol.*
- static const **DiscoveryConfigBuiltinPluginKindMask** **SPDP2** ()
  *Simple Participant **Discovery** (p. 1265) Protocol 2.0.*
- static const **DiscoveryConfigBuiltinPluginKindMask** **DPSE** ()
  Dynamic Participant discovery, Static Endpoint discovery.
- static const **DiscoveryConfigBuiltinPluginKindMask** **SDP** ()
  Simple discovery plugin (default).
- static const **DiscoveryConfigBuiltinPluginKindMask** **SDP2** ()
  Simple discovery plugin 2.0.

8.85.1 Detailed Description

<<***extension***>> *(p. 354)* A mask that selects the built-in discovery plugins to be used

8.85.2 Member Typedef Documentation

8.85.2.1 MaskType

```
typedef std::bitset<5>  rti::core::policy::DiscoveryConfigBuiltinPluginKindMask::MaskType
```

The base type, std::bitset.

8.85.3 Constructor & Destructor Documentation

8.85.3.1 DiscoveryConfigBuiltinPluginKindMask() [1/3]

```
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask::DiscoveryConfigBuiltinPluginKindMask ( )
[inline]
```

Creates an empty mask.

Referenced by **DPSE()**, **none()**, **SDP()**, **SDP2()**, **SEDP()**, **SPDP()**, and **SPDP2()**.

8.85.3.2 DiscoveryConfigBuiltinPluginKindMask() [2/3]

```
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask::DiscoveryConfigBuiltinPluginKindMask (
            uint64_t mask )  [inline], [explicit]
```

Creates a mask from the bits in an integer.

8.85.3.3 DiscoveryConfigBuiltinPluginKindMask() [3/3]

```
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask::DiscoveryConfigBuiltinPluginKindMask (
            const  MaskType & mask )  [inline]
```

Creates a mask from a std::bitset.

8.85.4 Member Function Documentation

8.85.4.1 none()

```
static const DiscoveryConfigBuiltinPluginKindMask rti::core::policy::DiscoveryConfigBuiltin↩
PluginKindMask::none ( ) [inline], [static]
```

No bits are set.

References **DiscoveryConfigBuiltinPluginKindMask()**.

8.85.4.2 SPDP()

```
static const DiscoveryConfigBuiltinPluginKindMask rti::core::policy::DiscoveryConfigBuiltin↩
PluginKindMask::SPDP ( ) [inline], [static]
```

Simple Participant **Discovery** (p. 1265) Protocol.

Enables the first phase of the Simple **Discovery** (p. 1265) Protocol (SDP), in which DomainParticipant's details are communicated to all other DomainParticipants in the same DDS domain by sending participant declaration messages, also known as participant DATA submessages or participant announcements.

References **DiscoveryConfigBuiltinPluginKindMask()**.

8.85.4.3 SEDP()

```
static const DiscoveryConfigBuiltinPluginKindMask rti::core::policy::DiscoveryConfigBuiltin↩
PluginKindMask::SEDP ( ) [inline], [static]
```

Simple Endpoint **Discovery** (p. 1265) Protocol.

Enables the second phase of the Simple **Discovery** (p. 1265) Protocol (SDP), in which the information (GUID, QoS, etc.) about your application's DataReaders and DataWriters is exchanged by sending publication/subscription declarations in DATA messages, also known as publication DATAs and subscription DATAs.

References **DiscoveryConfigBuiltinPluginKindMask()**.

8.85.4.4 SPDP2()

```
static const DiscoveryConfigBuiltinPluginKindMask rti::core::policy::DiscoveryConfigBuiltin↩
PluginKindMask::SPDP2 ( ) [inline], [static]
```

Simple Participant **Discovery** (p. 1265) Protocol 2.0.

Enables the Simple Participant **Discovery** (p. 1265) Protocol 2.0, in which a DomainParticipant's details are communicated to all other DomainParticipants in the same DDS domain by sending participant bootstrap messages. These bootstrap messages contain only a subset of the information in the Simple Participant **Discovery** (p. 1265) Protocol (SPDP) participant announcements that is required to match two participants and bootstrap the system. The DomainParticpant's full configuration is then sent reliably with participant configuration announcements. Two DomainParticipants that use SPDP2 will maintain liveliness using liveliness participant messages.

References **DiscoveryConfigBuiltinPluginKindMask()**.

8.85.4.5 DPSE()

```
static const DiscoveryConfigBuiltinPluginKindMask rti::core::policy::DiscoveryConfigBuiltin↩
PluginKindMask::DPSE ( ) [inline], [static]
```

Dynamic Participant discovery, Static Endpoint discovery.

Enables static endpoint discovery for a DomainParticipant. In this type of discovery, information from remote endpoints is extracted from a local DDS-XML file instead of being received over the network, reducing the number of exchanged packets and consequently reducing bandwidth consumption used for discovery. Using this value in **DiscoveryConfig↩ BuiltinPluginKindMask** (p. 1322) requires the 'librtilbedisc' library (included in the RTI Connext Professional bundles) to be reachable (PATH, LD_LIBRARY_PATH or DYLD_LIBRARY_PATH).

References **DiscoveryConfigBuiltinPluginKindMask()**.

8.85.4.6 SDP()

```
static const DiscoveryConfigBuiltinPluginKindMask rti::core::policy::DiscoveryConfigBuiltin↩
PluginKindMask::SDP ( ) [inline], [static]
```

Simple discovery plugin (default).

It is equivalent to SPDP + SEDP.

References **DiscoveryConfigBuiltinPluginKindMask()**.

8.85.4.7 SDP2()

```
static const DiscoveryConfigBuiltinPluginKindMask rti::core::policy::DiscoveryConfigBuiltin↩
PluginKindMask::SDP2 ( ) [inline], [static]
```

Simple discovery plugin 2.0.

It is equivalent to SPDP2 + SEDP.

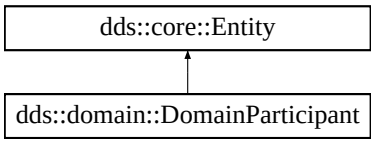
References **DiscoveryConfigBuiltinPluginKindMask()**.

8.86 dds::domain::DomainParticipant Class Reference

<<***reference-type***>> *(*p. *351)* Container for all **dds::core::Entity** (p. 1520) objects.

`#include <dds/domain/DomainParticipant.hpp>`

Inheritance diagram for dds::domain::DomainParticipant:



Public Types

- typedef **DomainParticipantListener** **Listener**

  *The **Listener** (*p. *1637) type that DomainParticipants use.*

Public Member Functions

- **DomainParticipant** (int32_t the_domain_id)

  *Create a new **DomainParticipant** (*p. *1326) with default Qos.*
- **DomainParticipant** (int32_t the_domain_id, const **dds::domain::qos::DomainParticipantQos** &the_↩qos, **dds::domain::DomainParticipantListener** ∗the_listener=NULL, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Create a new **DomainParticipant** (*p. *1326).*
- **DomainParticipant** (int32_t the_domain_id, const **dds::domain::qos::DomainParticipantQos** &the_qos, std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &mask= **dds::core::status↩::StatusMask::all**())

  *Create a new **DomainParticipant** (*p. *1326).*
- void **listener** (**Listener** ∗l, const **dds::core::status::StatusMask** &mask)

  *Register a listener with the **DomainParticipant** (*p. *1326).*
- **Listener** ∗ **listener** () const

  *Get the **DomainParticipant** (*p. *1326) listener.*

- void **set_listener** (std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &event_↩ mask)

  Sets the listener associated with this participant.
- void **set_listener** (std::shared_ptr< **Listener** > the_listener)

  Sets the listener associated with this participant.
- std::shared_ptr< **Listener** > **get_listener** () const

  Returns the listener currently associated with this participant.
- const **dds::domain::qos::DomainParticipantQos** **qos** () const

  *Gets the current QoS policies of this **DomainParticipant** (p. 1326).*
- void **qos** (const **dds::domain::qos::DomainParticipantQos** &the_qos)

  *Set the DomainParticipantQos setting for this **DomainParticipant** (p. 1326) instance.*
- **DomainParticipant** & **operator**<< (const **dds::domain::qos::DomainParticipantQos** &the_qos)

  Equivalent to set_qos(the_qos)
- const **DomainParticipant** & **operator**>> (**dds::domain::qos::DomainParticipantQos** &the_qos) const

  Equivalent to `get_qos()`
- int32_t **domain_id** () const

  Get the unique domain identifier.
- void **assert_liveliness** ()

  *Manually assert the liveliness of the **DomainParticipant** (p. 1326).*
- void **property** (const std::string &property_name, const std::string &value, bool propagate)

  *Set the value for a property that applies to a **DomainParticipant** (p. 1326).*
- bool **contains_entity** (const **dds::core::InstanceHandle** &a_handle)

  *Check whether or not the given handle represents an Entity that was created from the **DomainParticipant** (p. 1326).*
- **dds::core::Time** **current_time** ()

  Get the current time.
- **dds::pub::qos::PublisherQos** **default_publisher_qos** () const

  *Get the current default **dds::pub::qos::PublisherQos** (p. 2007).*
- **DomainParticipant** & **default_publisher_qos** (const **dds::pub::qos::PublisherQos** &the_qos)

  *Set the default **dds::pub::qos::PublisherQos** (p. 2007) for this **DomainParticipant** (p. 1326).*
- **dds::sub::qos::SubscriberQos** **default_subscriber_qos** () const

  *Get the current default **dds::sub::qos::SubscriberQos** (p. 2423).*
- **DomainParticipant** & **default_subscriber_qos** (const **dds::sub::qos::SubscriberQos** &the_qos)

  *Set the default **dds::sub::qos::SubscriberQos** (p. 2423) for this **DomainParticipant** (p. 1326).*
- **dds::topic::qos::TopicQos** **default_topic_qos** () const

  *Get the current default **dds::topic::qos::TopicQos** (p. 2509).*
- **DomainParticipant** & **default_topic_qos** (const **dds::topic::qos::TopicQos** &the_qos)

  *Set the default **dds::topic::qos::TopicQos** (p. 2509) for this **DomainParticipant** (p. 1326).*
- void **close_contained_entities** ()

  *<<**extension**>> (p. 354) Closes all the entities created from this **DomainParticipant** (p. 1326)*
- **dds::pub::qos::DataWriterQos** **default_datawriter_qos** () const

  *<<**extension**>> (p. 354) Get the current default **dds::pub::qos::DataWriterQos** (p. 1226).*
- void **default_datawriter_qos** (const **::dds::pub::qos::DataWriterQos** & **qos**)

  *<<**extension**>> (p. 354) Set the default **dds::pub::qos::DataWriterQos** (p. 1226) for this **DomainParticipant** (p. 1326).*
- **dds::sub::qos::DataReaderQos** **default_datareader_qos** () const

  *<<**extension**>> (p. 354) Get the current default **dds::sub::qos::DataReaderQos** (p. 1072).*
- void **default_datareader_qos** (const **::dds::sub::qos::DataReaderQos** & **qos**)

<<**extension**>> (p. 354) *Set the default* **dds::sub::qos::DataReaderQos** (p. 1072) *for this* **DomainParticipant** (p. 1326).

- void **register_contentfilter** (const **rti::topic::CustomFilter**< rti::topic::ContentFilterBase > &custom_filter, const std::string &filter_name)

  <<**extension**>> (p. 354) *Register a content filter which can be used to create a* **dds::topic::ContentFilteredTopic** (p. 954).
- void **unregister_contentfilter** (const std::string &filter_name)

  <<**extension**>> (p. 354) *Unregister a content filter previously registered with* **dds::domain::DomainParticipant::register_contentfilter(const rti::topic::CustomFilter<rti::topic::ContentFilterBase>& custom_filter, const std::string& filter_name)** (p. 1349).
- void **unregister_type** (const std::string &name)

  <<**extension**>> (p. 354) *Unregister a type that has previously been registered to this* **dds::domain::DomainParticipant** (p. 1326).
- bool **is_type_registered** (const std::string &name) const

  <<**extension**>> (p. 354) *Check if a type has previously been registered to this* **dds::domain::DomainParticipant** (p. 1326).
- void **add_peer** (const std::string &peer_descr_string)

  <<**extension**>> (p. 354) *Attempt to contact one or more additional peer participants.*
- void **remove_peer** (const std::string &peer_descr_string)

  <<**extension**>> (p. 354) *Remove one or more peer participants from the list of peers with which this* **dds::domain::DomainParticipant** (p. 1326) *will try to communicate.*
- **dds::core::Duration dns_tracker_polling_period** () const

  Retrieves the frequency used by the DNS tracker thread to query the DNS service.
- void **dns_tracker_polling_period** (const **dds::core::Duration** &polling_period)

  Configures the frequency in which the DNS tracker queries the DNS service.
- void **resume_endpoint_discovery** (const **dds::core::InstanceHandle** &remote_participant_handle)

  <<**extension**>> (p. 354) *Initiates endpoint discovery with the remote* **dds::domain::DomainParticipant** (p. 1326) *identified by its InstanceHandle.*
- void **delete_durable_subscription** (const **rti::core::EndpointGroup** &group)

  <<**extension**>> (p. 354) *Deletes an existing Durable Subscription on all Persistence Services.*
- void **register_durable_subscription** (const **rti::core::EndpointGroup** &group, const std::string &topic_name)

  <<**extension**>> (p. 354) *Registers a Durable Subscription on the specified* **dds::topic::Topic** (p. 2474) *on all Persistence Services.*
- **rti::core::status::DomainParticipantProtocolStatus participant_protocol_status** ()

  <<**extension**>> (p. 354) *Get the protocol status for this participant*
- rti::core::int64 **time_until_license_expiration** () const

  <<**extension**>> (p. 354) *Retrieve the remaining time until the license validated for this* **DomainParticipant** (p. 1326) *expires, with seconds precision.*

Static Public Member Functions

- static void **participant_factory_qos** (const **dds::domain::qos::DomainParticipantFactoryQos** & **qos**)

  Set the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378).
- static **dds::domain::qos::DomainParticipantFactoryQos participant_factory_qos** ()

  Get the current **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378).
- static void **finalize_participant_factory** ()

  Finalize the DomainParticipantFactory.
- static **dds::domain::qos::DomainParticipantQos default_participant_qos** ()

  Get the current DomainParticipantQos for this instance.
- static void **default_participant_qos** (const **dds::domain::qos::DomainParticipantQos** & **qos**)

  Set the DomainParticipantQos.

Related Functions

(Note that these are not member functions.)

- void **ignore** (const **dds::domain::DomainParticipant** &participant, const **dds::core::InstanceHandle** &handle)

  *Instructs RTI Connext to locally ignore a remote **dds::domain::DomainParticipant*** (p. *1326*).
- template<typename FwdIterator >
  void **ignore** (const **dds::domain::DomainParticipant** &participant, FwdIterator begin, FwdIterator end)

  *Instructs RTI Connext to locally ignore a group of remote **dds::domain::DomainParticipant*** (p. *1326*).
- **dds::core::InstanceHandleSeq discovered_participants** (const **dds::domain::DomainParticipant** &participant)

  Retrieves the list of other participants discovered by this participant.
- template<typename FwdIterator >
  FwdIterator **discovered_participants** (const **dds::domain::DomainParticipant** &participant, FwdIterator begin, FwdIterator end)

  Retrieves the list of other participants discovered by this participant.
- **dds::topic::ParticipantBuiltinTopicData discovered_participant_data** (const **dds::domain::DomainParticipant** &participant, const **dds::core::InstanceHandle** &participant_handle)

  Retrieves the information about one participant discovered by this participant.
- **DomainParticipant find** (int32_t **domain_id**)

  *Locates an existing **dds::domain::DomainParticipant*** (p. *1326*).
- void **banish_ignored_participants** (const **dds::domain::DomainParticipant** &participant)

  *<<**extension**>> (p. 354) Prevents ignored remote DomainParticipants from receiving traffic from the local **DomainParticipant*** (p. *1326*).
- **rti::core::optional_value**< std::string > **discovered_participant_subject_name** (const **dds::domain::DomainParticipant** &participant, const **dds::core::InstanceHandle** &participant_handle)

  *<<**extension**>> (p. 354) Returns **rti::core::policy::EntityName::name*** (p. *1531*) *for the specified **DomainParticipant*** (p. *1326*).
- **dds::core::InstanceHandleSeq discovered_participants_from_subject_name** (const **dds::domain::DomainParticipant** &participant, const **rti::core::optional_value**< std::string > &subject_name)

  *<<**extension**>> (p. 354) Returns a list of discovered **DomainParticipant*** (p. *1326*) *entities that have the given **rti::core::policy::EntityName::name*** (p. *1531*).
- **dds::domain::DomainParticipant find_participant_by_name** (const std::string &participant_name)

  *<<**extension**>> (p. 354) Locates an existing **dds::domain::DomainParticipant*** (p. *1326*) *by name.*
- template<typename ParticipantFwdIterator >
  uint32_t **find_participants** (ParticipantFwdIterator begin, uint32_t max_size)

  *<<**extension**>> (p. 354) Retrieves all the participants created by the application up to a maximum number.*
- template<typename ParticipantFwdIterator >
  uint32_t **find_participants** (ParticipantFwdIterator begin)

  *<<**extension**>> (p. 354) Retrieves all the participants created by the application.*
- const **dds::core::xtypes::DynamicType** & **find_type** (const **dds::domain::DomainParticipant** &participant, const std::string &type_name)

  *<<**extension**>> (p. 354) Retrieves a type registered with this participant*
- void **register_type** (**dds::domain::DomainParticipant** &participant, const std::string &name, const **dds::core::xtypes::DynamicType** &type, const **rti::core::xtypes::DynamicDataTypeSerializationProperty** &serialization_property= **rti::core::xtypes::DynamicDataTypeSerializationProperty::DEFAULT**)

  *<<**extension**>> (p. 354) Registers a DynamicType with specific serialization properties*

- template<typename T >
  void **register_type** (const std::string &registered_type_name= **dds::topic::topic_type_name**< T >::value())

  <<***extension***>> *(p. 354) Registers a User-Generated Type with RTI Connext. This function is used along with XML Application Creation.*

8.86.1 Detailed Description

<<***reference-type***>> *(p. 351)* Container for all **dds::core::Entity** (p. 1520) objects.

- It acts as a container for all other **dds::core::Entity** (p. 1520) objects.
- It acts as a *factory* for the **dds::pub::Publisher** (p. 1993), **dds::sub::Subscriber** (p. 2410), **dds::topic::Topic** (p. 2474) and MultiTopic **dds::core::Entity** (p. 1520) objects.
- It represents the participation of the application on a communication plane that isolates applications running on the same set of physical computers from each other. A domain establishes a virtual network linking all applications that share the same `domainId` and isolating them from applications running on different domains. In this way, several independent distributed applications can coexist in the same physical network without interfering, or even being aware of each other.
- It provides administration services in the domain, offering operations that allow the application to ignore locally any information about a given participant (**dds::domain::ignore** (p. 635)), publication (**dds::pub::ignore** (p. 647)), subscription (**dds::sub::ignore()** (p. 667)) or topic (**dds::topic::ignore()** (p. 693)()).

The following operations may be called even if the **dds::domain::DomainParticipant** (p. 1326) is not enabled. Operations NOT in this list will fail with **dds::core::NotEnabledError** (p. 1869) \ if called on a disabled **DomainParticipant** (p. 1326).

- **dds::core::Entity::enable** (p. 1524),
- **dds::domain::DomainParticipant::qos(const dds::domain::qos::DomainParticipantQos&)** (p. 1337), **dds::domain::DomainParticipant::qos() const** (p. 1336),
- **dds::domain::DomainParticipant::set_listener** (p. 1335), **dds::domain::DomainParticipant::get_listener** (p. 1336),
- The constructors of **rti::pub::FlowController** (p. 1572), **dds::topic::Topic** (p. 2474), **dds::pub::Publisher** (p. 1993), **dds::sub::Subscriber** (p. 2410).
- **dds::domain::DomainParticipant::default_topic_qos** (p. 1344), **dds::domain::DomainParticipant::default_topic_qos()** (p. 1344), **dds::domain::DomainParticipant::default_publisher_qos** (p. 1342), **dds::domain::DomainParticipant::default_publisher_qos()** (p. 1342), **dds::domain::DomainParticipant::default_subscriber_qos(const dds::sub::qos::SubscriberQos & qos)** (p. 1343), **dds::domain::DomainParticipant::default_subscriber_qos** (p. 1343), **dds::domain::DomainParticipant::default_datareader_qos** (p. 1348), **dds::domain::DomainParticipant::default_datareader_qos()** (p. 1348), **dds::domain::DomainParticipant::default_datawriter_qos** (p. 1346), **dds::domain::DomainParticipant::default_datawriter_qos()** (p. 1346), ;
- Operations for looking up topics: **dds::topic::find()** (p. 694);
- Operations that access status: **dds::core::cond::StatusCondition::StatusCondition(const dds::core::Entity&)** (p. 2373), **dds::core::Entity::status_changes()** (p. 1525).

QoS:

dds::domain::qos::DomainParticipantQos (p. 1384)

Status:

Status Kinds (p. 429)

Listener:

DomainParticipantListener (p. 1382)

See also

Operations Allowed in Listener Callbacks (p. 1639)

Other operations related to a **DomainParticipant** (p. 1326) in other namespaces:

- Related to **dds::topic::Topic** (p. 2474): **dds::topic::find()** (p. 694), **dds::topic::discover_any_topic** (p. 690), **dds::topic::discover_topic_data** (p. 691), **dds::topic::ignore()** (p. 693) and similar operations
- Related to **dds::sub::Subscriber** (p. 2410): **rti::sub::find_subscribers()** (p. 757), **dds::sub::builtin_↩ subscriber()** (p. 671), **rti::sub::implicit_subscriber()** (p. 754)
- Related to **dds::sub::DataReader** (p. 976): **dds::sub::ignore()** (p. 667), **rti::sub::find_datareader_by_↩ name()** (p. 762)
- Related to **dds::pub::Publisher** (p. 1993): **rti::pub::find_publishers()** (p. 743), **rti::pub::implicit_↩ publisher()** (p. 738)
- Related to **dds::pub::DataWriter** (p. 1138): **dds::pub::ignore()** (p. 647), **rti::pub::find_datawriter_by_↩ name()** (p. 748)
- Related to **rti::pub::FlowController** (p. 1572): **rti::pub::find_flow_controller()** (p. 750)

dds::core::QosProvider::create_participant_from_config (p. 2046) to create a **DomainParticipant** (p. 1326) from an XML definition

Participant Use Cases (p. 306)

Entity Use Cases (p. 325)

Examples

Foo.hpp, **Foo_publisher.cxx**, and **Foo_subscriber.cxx**.

8.86.2 Member Typedef Documentation

8.86.2.1 Listener

```
typedef DomainParticipantListener dds::domain::DomainParticipant::Listener
```

The **Listener** (p. 1637) type that DomainParticipants use.

8.86.3 Constructor & Destructor Documentation

8.86.3.1 DomainParticipant() [1/3]

```
dds::domain::DomainParticipant::DomainParticipant (
            int32_t the_domain_id )  [inline], [explicit]
```

Create a new **DomainParticipant** (p. 1326) with default Qos.

Same as **DomainParticipant(int32_t, const dds::domain::qos::DomainParticipantQos&, dds::domain::Domain↩ ParticipantListener∗, const dds::core::status::StatusMask&)** (p. 1332) except that it uses **the default Domain↩ ParticipantQos** (p. 1522)

8.86.3.2 DomainParticipant() [2/3]

```
dds::domain::DomainParticipant::DomainParticipant (
            int32_t the_domain_id,
            const  dds::domain::qos::DomainParticipantQos & the_qos,
             dds::domain::DomainParticipantListener * the_listener = NULL,
            const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a new **DomainParticipant** (p. 1326).

[DEPRECATED] When using a listener, prefer the constructor that receives a shared_ptr<**Listener** (p. 1637)> instead of a regular Listener* pointer.

The **DomainParticipant** (p. 1326) signifies that the application intends to join the Domain identified by the domain_id argument.

Parameters

the_domain↩ _id	the id of the domain joined by this **DomainParticipant** (p. 1326). **[range]** [>=0], and does not violate guidelines stated in **rti::core::RtpsWellKnownPorts** (p. 2258).
the_qos	The QoS settings for this **DomainParticipant** (p. 1326). See also set_qos().
the_listener	The listener (NULL by default)
mask	Changes of communication status to be invoked on the listener.

Precondition

The specified QoS policies must be consistent or the operation will fail and no **dds::domain::DomainParticipant** (p. 1326) will be created.

If you want to create multiple participants on a given host in the same domain, make sure each one has a different participant index (set in the **rti::core::policy::WireProtocol** (p. 2630)). This in turn will ensure each participant uses a different port number (since the unicast port numbers are calculated from the participant index and the domain ID).

Note that if there is a single participant per host in a given domain, the participant index can be left at the default value (-1).

MT Safety:

Unsafe. On VxWorks, none of the following is safe until the first **dds::domain::DomainParticipant** (p. 1326) has been created or **dds::core::QosProvider::Default()** (p. 2036) has been obtained for the first time:

- creating several **dds::domain::DomainParticipant** (p. 1326) in parallel;
- calling **dds::core::QosProvider::Default()** (p. 2036) from two or more threads in parallel;
- creating a **dds::domain::DomainParticipant** (p. 1326) and calling **dds::core::QosProvider::Default()** (p. 2036) in parallel.

See also

Specifying QoS on entities (p. 513) for information on setting QoS before entity creation

dds::domain::qos::DomainParticipantQos (p. 1384) for rules on consistency among QoS

NDDS_DISCOVERY_PEERS (p. 559)

dds::core::QosProvider (p. 2025)

dds::domain::DomainParticipant::default_participant_qos() (p. 1341)

dds::domain::DomainParticipant::set_listener (p. 1335)

8.86.3.3 DomainParticipant() [3/3]

```
dds::domain::DomainParticipant::DomainParticipant (
             int32_t the_domain_id,
             const  dds::domain::qos::DomainParticipantQos & the_qos,
             std::shared_ptr<  Listener > the_listener,
             const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a new **DomainParticipant** (p. 1326).

The **DomainParticipant** (p. 1326) signifies that the application intends to join the Domain identified by the domain_id argument.

Parameters

the_domain_id	the id of the domain joined by this **DomainParticipant** (p. 1326). **[range]** [>=0], and does not violate guidelines stated in **rti::core::RtpsWellKnownPorts** (p. 2258).
the_qos	The QoS settings for this **DomainParticipant** (p. 1326). See also set_qos().
the_listener	A shared_ptr to the listener. See **set_listener()** (p. 1335) for more information.
mask	Changes of communication status to be invoked on the listener.

Precondition

The specified QoS policies must be consistent or the operation will fail and no **dds::domain::DomainParticipant** (p. 1326) will be created.

If you want to create multiple participants on a given host in the same domain, make sure each one has a different participant index (set in the **rti::core::policy::WireProtocol** (p. 2630)). This in turn will ensure each participant uses a different port number (since the unicast port numbers are calculated from the participant index and the domain ID).

Note that if there is a single participant per host in a given domain, the participant index can be left at the default value (-1).

MT Safety:

Unsafe. On VxWorks, none of the following is safe until the first **dds::domain::DomainParticipant** (p. 1326) has been created or **dds::core::QosProvider::Default()** (p. 2036) has been obtained for the first time:

- creating several **dds::domain::DomainParticipant** (p. 1326) in parallel;
- calling **dds::core::QosProvider::Default()** (p. 2036) from two or more threads in parallel;
- creating a **dds::domain::DomainParticipant** (p. 1326) and calling **dds::core::QosProvider::Default()** (p. 2036) in parallel.

See also

Specifying QoS on entities (p. 513) for information on setting QoS before entity creation

dds::domain::qos::DomainParticipantQos (p. 1384) for rules on consistency among QoS

NDDS_DISCOVERY_PEERS (p. 559)

dds::core::QosProvider (p. 2025)

dds::domain::DomainParticipant::default_participant_qos() (p. 1341)

dds::domain::DomainParticipant::set_listener (p. 1335)

8.86.4 Member Function Documentation

8.86.4.1 listener() [1/2]

```
void dds::domain::DomainParticipant::listener (
              Listener * l,
             const  dds::core::status::StatusMask & mask )  [inline]
```

Register a listener with the **DomainParticipant** (p. 1326).

[DEPRECATED] The use of **set_listener()** (p. 1335) is recommended. Unlike this function, set_listener receives a shared_ptr which simplifies the management of listener's lifecycle.

8.86.4.2 listener() [2/2]

```
Listener * dds::domain::DomainParticipant::listener ( ) const  [inline]
```

Get the **DomainParticipant** (p. 1326) listener.

[DEPRECATED] Prefer **get_listener()** (p. 1336) instead of this function.

Returns

The existing listener attached to the **DomainParticipant** (p. 1326) or NULL if no listener has been set.

8.86.4.3 set_listener() [1/2]

```
void dds::domain::DomainParticipant::set_listener (
            std::shared_ptr<  Listener > the_listener,
            const  dds::core::status::StatusMask & event_mask )  [inline]
```

Sets the listener associated with this participant.

The **DomainParticipant** (p. 1326) will hold the `shared_ptr`, ensuring that the listener is not deleted at least until this **DomainParticipant** (p. 1326) is deleted or the listener is reset with `set_listener(nullptr)`.

Warning

It's recommended that the listener implementation doesn't hold a permanent reference to the participant. If it does, the application needs to manually reset the listener or manually close this participant to ensure that there is no cycle that prevents the destruction of these two objects.

Parameters

the_listener	A shared pointer to the listener to receive status updates or `nullptr` to reset the current listener and stop receiving status updates.
event_mask	A mask that indicates which status updates will be notified to the listener

MT Safety:

Unsafe. This method is not synchronized with the listener callbacks, so it is possible to set a new listener on a participant when the old listener is in a callback.

Care should therefore be taken not to delete any listener that has been set on an enabled participant unless some application-specific means are available of ensuring that the old listener cannot still be in use.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

set_listener (abstract) (p. 1523)

8.86.4.4 set_listener() [2/2]

```
void dds::domain::DomainParticipant::set_listener (
            std::shared_ptr< Listener > the_listener ) [inline]
```

Sets the listener associated with this participant.

If `the_listener` is not `nullptr`, this overload is equivalent to:
```
participant.set_listener(the_listener,
dds::core::status::StatusMask::all());
```

If `the_listener` is `nullptr`, it is equivalent to:
```
participant.set_listener(nullptr, dds::core::status::StatusMask::none());
```

Parameters

the_listener	A shared pointer to the listener to receive status updates or `nullptr` to reset the current listener and stop receiving status updates.

8.86.4.5 get_listener()

```
std::shared_ptr< Listener > dds::domain::DomainParticipant::get_listener ( ) const [inline]
```

Returns the listener currently associated with this participant.

Returns

The shared pointer to the current listener or `nullptr` if there is currently no listener associated to this entity.

8.86.4.6 qos() [1/2]

```
const  dds::domain::qos::DomainParticipantQos dds::domain::DomainParticipant::qos ( ) const  [inline]
```

Gets the current QoS policies of this **DomainParticipant** (p. 1326).

Returns

The current QoS policies.

8.86.4.7 qos() [2/2]

```
void dds::domain::DomainParticipant::qos (
            const  dds::domain::qos::DomainParticipantQos & the_qos )  [inline]
```

Set the DomainParticipantQos setting for this **DomainParticipant** (p. 1326) instance.

Parameters

the_qos	Set of policies to be applied to the **DomainParticipant** (p. 1326). Immutable policies cannot be changed after this entity has been enabled.

Exceptions

dds::core::ImmutablePolicyError (p. *1609*)	if `the_qos` contains a different value for an immutable policy.
dds::core::InconsistentPolicyError (p. *1610*)	if some policies are inconsistent.
dds::core::NotAllowedBySecurityError (p. *1868*)	if the security extensions do not allow some policies.

See also

dds::domain::qos::DomainParticipantQos (p. 1384) for rules on consistency among QoS policies

set_qos (abstract) (p. 1522)

default_participant_qos() (p. 1341)

dds::core::QosProvider::participant_qos() (p. 2031)

8.86.4.8 operator<<()

```
DomainParticipant & dds::domain::DomainParticipant::operator<< (
            const  dds::domain::qos::DomainParticipantQos & the_qos )  [inline]
```

Equivalent to set_qos(the_qos)

8.86.4.9 operator>>()

```
const DomainParticipant & dds::domain::DomainParticipant::operator>> (
            dds::domain::qos::DomainParticipantQos & the_qos ) const [inline]
```

Equivalent to `get_qos()`

Parameters

the_qos	the new qos for this **DomainParticipant** (p. 1326).

8.86.4.10 domain_id()

```
int32_t dds::domain::DomainParticipant::domain_id ( ) const [inline]
```

Get the unique domain identifier.

This operation retrieves the domain id used to create the **dds::domain::DomainParticipant** (p. 1326). The domain id identifies the DDS domain to which the **dds::domain::DomainParticipant** (p. 1326) belongs. Each DDS domain represents a separate data 'communication plane' isolated from other domains.

Returns

the domain id

8.86.4.11 assert_liveliness()

```
void dds::domain::DomainParticipant::assert_liveliness ( ) [inline]
```

Manually assert the liveliness of the **DomainParticipant** (p. 1326).

This is used in combination with the **dds::core::policy::Liveliness** (p. 1645) to indicate to RTI Connext that the entity remains active.

You need to use this operation if the **dds::domain::DomainParticipant** (p. 1326) contains **dds::pub::DataWriter** (p. 1138) entities with the **dds::core::policy::Liveliness::kind** (p. 1649) set to **dds::core::policy::LivelinessKind↩ _def::MANUAL_BY_PARTICIPANT** (p. 1655) and it only affects the liveliness of those **dds::pub::DataWriter** (p. 1138) entities. Otherwise, it has no effect.

Note: writing data via the **dds::pub::DataWriter::write()** (p. 1148) or **dds::pub::DataWriter::write(const T&,const dds::core::Time&)** (p. 1149) operation asserts liveliness on the **dds::pub::DataWriter** (p. 1138) itself and its **dds↩ ::domain::DomainParticipant** (p. 1326). Consequently the use of **assert_liveliness()** (p. 1338) is only needed if the application is not writing data regularly.

8.86.4.12 property()

```
void dds::domain::DomainParticipant::property (
            const std::string & property_name,
            const std::string & value,
            bool propagate )  [inline]
```

Set the value for a property that applies to a **DomainParticipant** (p. 1326).

Warning

This method is not implemented in all APIs and it's intended only for testing purposes. You should use **dds::domain::DomainParticipant::qos(const dds::domain::qos::DomainParticipantQos&)** (p. 1337) instead.

Parameters

property_name	*<<**in**>> (p. 355).* Name of the property that you want to set.
value	*<<**in**>> (p. 355).* New value for the property.
propagate	*<<**in**>> (p. 355).* Indicates if the property will be propagated or not.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

dds::pub::DataWriter::set_property

dds::sub::DataReader::set_property

dds::domain::DomainParticipant::qos(const dds::domain::qos::DomainParticipantQos&) (p. 1337)

8.86.4.13 contains_entity()

```
bool dds::domain::DomainParticipant::contains_entity (
            const  dds::core::InstanceHandle & a_handle )  [inline]
```

Check whether or not the given handle represents an Entity that was created from the **DomainParticipant** (p. 1326).

This operation checks whether or not the given `a_handle` represents an **dds::core::Entity** (p. 1520) that was created from the **dds::domain::DomainParticipant** (p. 1326). The containment applies recursively. That is, it applies both to entities (TopicDescription, **dds::pub::Publisher** (p. 1993), or **dds::sub::Subscriber** (p. 2410)) created directly using the **dds::domain::DomainParticipant** (p. 1326) as well as entities created using a contained **dds::pub::Publisher** (p. 1993), or **dds::sub::Subscriber** (p. 2410) as the factory, and so forth.

The `instance` handle for an **dds::core::Entity** (p. 1520) may be obtained from built-in topic data, from various statuses, or from the operation **dds::core::Entity::instance_handle()** (p. 1525).

Parameters

a_handle	<<*in*>> *(p. 355)* **dds::core::InstanceHandle** (p. 1612) of the **dds::core::Entity** (p. 1520) to be checked.

Returns

true if **dds::core::Entity** (p. 1520) is contained by the **dds::domain::DomainParticipant** (p. 1326), or false otherwise.

8.86.4.14 current_time()

```
dds::core::Time dds::domain::DomainParticipant::current_time ( )  [inline]
```

Get the current time.

The current value of the time that RTI Connext uses to time-stamp **dds::pub::DataWriter** (p. 1138) and to set the reception-timestamp for the data updates that it receives.

Exceptions

One	of the **Standard Exceptions** (p. 428)

Returns

The current time

8.86.4.15 participant_factory_qos() [1/2]

```
static void dds::domain::DomainParticipant::participant_factory_qos (
            const  dds::domain::qos::DomainParticipantFactoryQos & qos )  [inline], [static]
```

Set the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378).

Parameters

qos	The DomainParticipantFactoryQos to set.

8.86.4.16 participant_factory_qos() [2/2]

```
static dds::domain::qos::DomainParticipantFactoryQos dds::domain::DomainParticipant::participant↩
_factory_qos ( ) [inline], [static]
```

Get the current **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378).

Returns

The current DomainParticipantFactoryQos

8.86.4.17 finalize_participant_factory()

```
static void dds::domain::DomainParticipant::finalize_participant_factory ( ) [inline], [static]
```

Finalize the DomainParticipantFactory.

The DomainParticipantFactory is a singleton that the C++ API uses implicitly to create participants. RTI Connext provides this operation for users who want to release memory used by the participant factory.

Warning

This method uses a static variable. To avoid undefined behavior in the order of destruction it shouldn't be called in the destructor of a type for which other global or static variables exist.

Examples

Foo_publisher.cxx, and **Foo_subscriber.cxx**.

8.86.4.18 default_participant_qos() [1/2]

```
static dds::domain::qos::DomainParticipantQos dds::domain::DomainParticipant::default_participant↩
_qos ( ) [inline], [static]
```

Get the current DomainParticipantQos for this instance.

Returns

The current DomainParticipantQos

See also

Default QoS (p. 1522)

8.86.4.19 default_participant_qos() [2/2]

```
static void dds::domain::DomainParticipant::default_participant_qos (
            const dds::domain::qos::DomainParticipantQos & qos ) [inline], [static]
```

Set the DomainParticipantQos.

The UserData and EntityFactory policies can be changed. The other policies are immutable.

Parameters

qos	The DomainParticipantQos to set.

See also

Default QoS (p. 1522)

8.86.4.20 default_publisher_qos() [1/2]

```
dds::pub::qos::PublisherQos dds::domain::DomainParticipant::default_publisher_qos ( ) const  [inline]
```

Get the current default **dds::pub::qos::PublisherQos** (p. 2007).

The retrieved qos will match the set of values specified on the last successful call to **default_publisher_qos(const dds::pub::qos::PublisherQos& qos)** (p. 1342), or else, if the call was never made, the RTI Connext default values for the **dds::pub::qos::PublisherQos** (p. 2007).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

If no PublisherQos is specified when a **dds::pub::Publisher** (p. 1993) is constructed, the default value of the QoS set in the factory, equivalent to the value obtained by calling this method, will be used to create the Publisher.

MT Safety:

UNSAFE. It is not safe to retrieve the default publisher QoS from a **DomainParticipant** (p. 1326) while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_publisher_qos** (p. 1342)

Returns

The current default PublisherQos

8.86.4.21 default_publisher_qos() [2/2]

```
DomainParticipant & dds::domain::DomainParticipant::default_publisher_qos (
            const  dds::pub::qos::PublisherQos & the_qos )  [inline]
```

Set the default **dds::pub::qos::PublisherQos** (p. 2007) for this **DomainParticipant** (p. 1326).

This set of default values will be used for a newly created **dds::pub::Publisher** (p. 1993) if no **dds::pub::qos::PublisherQos** (p. 2007) is specified when constructing the Publisher.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **dds::core::InconsistentPolicyError** (p. 1610)

MT Safety:

UNSAFE. It is not safe to set the default publisher QoS for a **DomainParticipant** (p. 1326) while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_publisher_qos** (p. 1342), **dds::domain::DomainParticipant::default_publisher_qos()** (p. 1342) or creating a Publisher with default QoS values.

Parameters

the_qos	The PublisherQos to set.

Returns

This **DomainParticipant** (p. 1326) instance

Exceptions

One	of the standard **Exceptions** (p. 427), or **dds::core::InconsistentPolicyError** (p. 1610)

8.86.4.22 default_subscriber_qos() [1/2]

dds::sub::qos::SubscriberQos dds::domain::DomainParticipant::default_subscriber_qos ( ) const [inline]

Get the current default **dds::sub::qos::SubscriberQos** (p. 2423).

The retrieved qos will match the set of values specified on the last successful call to **dds::domain::DomainParticipant::default_subscriber_qos(const dds::sub::qos::SubscriberQos & qos)** (p. 1343), or dds::domain::DomainParticipant::set_default_subscriber_qos_with_profile, or else, if the call was never made, the default values listed in **dds::sub::qos::SubscriberQos** (p. 2423).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

If no SubscriberQos is specified when a **dds::sub::Subscriber** (p. 2410) is constructed, the default value of the QoS set in the factory, equivalent to the value obtained by calling this method, will be used to create the Subscriber.

MT Safety:

UNSAFE. It is not safe to retrieve the default Subscriber QoS from a **DomainParticipant** (p. 1326) while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_subscriber_qos(const dds::sub::qos::SubscriberQos & qos)** (p. 1343).

See also

dds::sub::Subscriber::Subscriber (p. 2413)

Returns

The current default SubscriberQos

8.86.4.23 default_subscriber_qos() [2/2]

```
DomainParticipant & dds::domain::DomainParticipant::default_subscriber_qos (
            const  dds::sub::qos::SubscriberQos & the_qos )  [inline]
```

Set the default **dds::sub::qos::SubscriberQos** (p. 2423) for this **DomainParticipant** (p. 1326).

This set of default values will be used for a newly created **dds::sub::Subscriber** (p. 2410) when the qos parameter is not specified

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **dds::core::InconsistentPolicyError** (p. 1610)

MT Safety:

UNSAFE. It is not safe to set the default Subscriber QoS for a **DomainParticipant** (p. 1326) while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_subscriber_qos(const dds::sub::qos::SubscriberQos & qos)** (p. 1343), **dds::domain::DomainParticipant::default_subscriber_qos** (p. 1343) or calling **dds::sub::Subscriber::Subscriber** (p. 2413) with SUBSCRIBER_QOS_DEFAULT as the `qos` parameter.

Parameters

the_qos	The SubscriberQos to set.

Exceptions

One	of the standard **Exceptions** (p. 427), or **dds::core::InconsistentPolicyError** (p. 1610)

8.86.4.24 default_topic_qos() [1/2]

```
dds::topic::qos::TopicQos dds::domain::DomainParticipant::default_topic_qos ( ) const  [inline]
```

Get the current default **dds::topic::qos::TopicQos** (p. 2509).

The retrieve d qos will match the set of values specified on the last successful call to **default_topic_qos(const dds::topic::qos::TopicQos& qos)** (p. 1345), or else, if the call was never made, the RTI Connext default values for the **dds::topic::qos::TopicQos** (p. 2509).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

If no TopicQos is specified when a **dds::topic::Topic** (p. 2474) is constructed, the default value of the QoS set in the factory, equivalent to the value obtained by calling this method, will be used to create the Topic.

MT Safety:

UNSAFE. It is not safe to retrieve the default Topic QoS from a **DomainParticipant** (p. 1326) while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_topic_qos** (p. 1344)

Returns

The current default TopicQos

8.86.4.25 default_topic_qos() [2/2]

```
DomainParticipant & dds::domain::DomainParticipant::default_topic_qos (
            const  dds::topic::qos::TopicQos & the_qos )  [inline]
```

Set the default **dds::topic::qos::TopicQos** (p. 2509) for this **DomainParticipant** (p. 1326).

This set of default values will be used for a newly created **dds::topic::Topic** (p. 2474) if no **dds::topic::qos::TopicQos** (p. 2509) is specified when constructing the Topic.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **dds::core::InconsistentPolicyError** (p. 1610)

MT Safety:

UNSAFE. It is not safe to set the default topic QoS for a **DomainParticipant** (p. 1326) while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_topic_qos** (p. 1344), **dds::domain::DomainParticipant::default_topic_qos()** (p. 1344) or creating a Topic with default QoS values.

Parameters

the_qos	The TopicQos to set.

Returns

This **DomainParticipant** (p. 1326) instance

Exceptions

One	of the standard **Exceptions** (p. 427), or **dds::core::InconsistentPolicyError** (p. 1610)

8.86.4.26 close_contained_entities()

```
void close_contained_entities ( )
```

<<***extension***>> *(p. 354)* Closes all the entities created from this **DomainParticipant** (p. 1326)

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Calling this function explicitly is not necessary to close a **DomainParticipant** (p. 1326).

This operation closes all contained **dds::pub::Publisher** (p. 1993) (including an implicit Publisher, if one exists), **dds::sub::Subscriber** (p. 2410) (including implicit Subscriber), **dds::topic::Topic** (p. 2474), **dds::topic::ContentFilteredTopic** (p. 954), and MultiTopic objects.

Prior to closing each contained entity, this operation will recursively call the corresponding **close_contained_entities()** (p. 1345) operation on each contained entity (if applicable). This pattern is applied recursively. In this manner the operation **close_contained_entities()** (p. 1345) on the **dds::domain::DomainParticipant** (p. 1326) will end up recursively closing all the entities contained in the **dds::domain::DomainParticipant** (p. 1326), including the **dds::pub::DataWriter** (p. 1138), **dds::sub::DataReader** (p. 976), as well as the **dds::sub::cond::QueryCondition** (p. 2059), **dds::sub::cond::ReadCondition** (p. 2133), and **rti::sub::TopicQuery** (p. 2516) objects belonging to the contained **dds::sub::DataReader** (p. 976).

The operation will fail with **dds::core::PreconditionNotMetError** (p. 1941) if any of the contained entities is in a state where it cannot be closed .

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::PreconditionNotMetError** (p. 1941).

8.86.4.27 default_datawriter_qos() [1/2]

```
dds::pub::qos::DataWriterQos default_datawriter_qos ( ) const
```

<<***extension***>> *(p. 354)* Get the current default **dds::pub::qos::DataWriterQos** (p. 1226).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The retrieved qos will match the set of values specified on the last successful call to **default_datawriter_qos(const ::dds::pub::qos::DataWriterQos& qos)** (p. 1347), or else, if the call was never made, the RTI Connext default values for the **dds::pub::qos::DataWriterQos** (p. 1226).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

If no DataWriterQos is specified when a **dds::pub::DataWriter** (p. 1138) is constructed, the default value of the QoS set in the factory, equivalent to the value obtained by calling this method, will be used to create the DataWriter.

MT Safety:

UNSAFE. It is not safe to retrieve the default DataWriterQoS from a DomainPartipant while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_datawriter_qos** (p. 1346).

Returns

The current default DataWriterQos

8.86.4.28 default_datawriter_qos() [2/2]

```
void default_datawriter_qos (
            const  ::dds::pub::qos::DataWriterQos & qos )
```

*<<**extension**>> (p. 354)* Set the default **dds::pub::qos::DataWriterQos** (p. 1226) for this **DomainParticipant** (p. 1326).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This set of default values will be used for a newly created **dds::pub::DataWriter** (p. 1138) if no **dds::pub::qos::DataWriterQos** (p. 1226) is specified when constructing the DataWriter.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **dds::core::InconsistentPolicyError** (p. 1610)

MT Safety:

UNSAFE. It is not safe to set the default DataWriter QoS for a **DomainParticipant** (p. 1326) while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_datawriter_qos** (p. 1346) or **dds::domain::DomainParticipant::default_datawriter_qos()** (p. 1346) or creating a DataWriter with default QoS values.

Parameters

qos	The DataWriterQos to set.

Exceptions

One	of the standard **Exceptions** (p. 427), or **dds::core::InconsistentPolicyError** (p. 1610)

8.86.4.29 default_datareader_qos() [1/2]

```
dds::sub::qos::DataReaderQos default_datareader_qos ( ) const
```

<<***extension***>> *(p. 354)* Get the current default **dds::sub::qos::DataReaderQos** (p. 1072).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The retrieved qos will match the set of values specified on the last successful call to **default_datareader_qos(const ::dds::sub::qos::DataReaderQos& qos)** (p. 1348), or else, if the call was never made, the RTI Connext default values for the **dds::sub::qos::DataReaderQos** (p. 1072).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

If no DataReaderQos is specified when a **dds::sub::DataReader** (p. 976) is constructed, the default value of the QoS set in the factory, equivalent to the value obtained by calling this method, will be used to create the DataReader.

MT Safety:

UNSAFE. It is not safe to retrieve the default DataReader QoS from a **DomainParticipant** (p. 1326) while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_datareader_qos** (p. 1348).

Returns

The current default DataReaderQos

8.86.4.30 default_datareader_qos() [2/2]

```
void default_datareader_qos (
            const  ::dds::sub::qos::DataReaderQos & qos )
```

<<***extension***>> *(p. 354)* Set the default **dds::sub::qos::DataReaderQos** (p. 1072) for this **DomainParticipant** (p. 1326).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This set of default values will be used for a newly created **dds::sub::DataReader** (p. 976) if no **dds::sub::qos::DataReaderQos** (p. 1072) is specified when constructing the DataReader.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **dds::core::InconsistentPolicyError** (p. 1610)

MT Safety:

UNSAFE. It is not safe to set the default DataReader QoS for a **DomainParticipant** (p. 1326) while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_datareader_qos** (p. 1348) or **dds::domain::DomainParticipant::default_datareader_qos()** (p. 1348). or creating a DataReader with default QoS values.

Parameters

qos	The DataReaderQos to set.

Exceptions

One	of the standard **Exceptions** (p. 427), or **dds::core::InconsistentPolicyError** (p. 1610)

8.86.4.31 register_contentfilter()

```
void register_contentfilter (
            const  rti::topic::CustomFilter< rti::topic::ContentFilterBase > & custom_filter,
            const std::string & filter_name )
```

<<***extension***>> *(p. 354)* Register a content filter which can be used to create a **dds::topic::ContentFilteredTopic** (p. 954).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

DDS specifies a SQL-like content filter for use by content filtered topics. If this filter does not meet your filtering requirements, you can register a custom filter.

To use a custom filter, it must be registered in the following places:

- In any application that uses the custom filter to create a **dds::topic::ContentFilteredTopic** (p. 954) and the corresponding **dds::sub::DataReader** (p. 976).
- In each application that writes the data to the applications mentioned above.

For example, suppose Application A on the subscription side creates a Topic named X and a ContentFilteredTopic named filteredX (and a corresponding DataReader), using a previously registered content filter, myFilter. With only that, you will have filtering at the subscription side. If you also want to perform filtering in any application that publishes Topic X, then you also need to register the same definition of the ContentFilter myFilter in that application.

Each `filter_name` can only be used to registered a content filter once with a **dds::domain::DomainParticipant** (p. 1326).

See also

dds::domain::DomainParticipant::unregister_contentfilter(const std::string & filter_name) (p. 1350)

Parameters

custom_filter	A custom filter
filter_name	Name of the filter. The name must be unique within the **DomainParticipant** (p. 1326) and must not exceed 255 characters.

Exceptions

One	of the standard **Exceptions** (p. 427)

See also

Creating Custom Content Filters (p. 333)

8.86.4.32 unregister_contentfilter()

```
void unregister_contentfilter (
            const std::string & filter_name )
```

<<***extension***>> *(p. 354)* Unregister a content filter previously registered with **dds::domain::DomainParticipant::register_contentfilter(const rti::topic::CustomFilter<rti::topic::ContentFilterBase>& custom_filter, const std::string& filter_name)** (p. 1349).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

A `filter_name` can be unregistered only if it has been previously registered to the **dds::domain::Domain↩ Participant** (p. 1326) with **dds::domain::DomainParticipant::register_contentfilter(const rti::topic::CustomFilter < rti::topic::ContentFilterBase > & custom_filter, const std::string & filter_name)** (p. 1349).

The unregistration of filter is not allowed if there are any existing **dds::topic::ContentFilteredTopic** (p. 954) objects that are using the filter. If the operation is called on a filter with existing **dds::topic::ContentFilteredTopic** (p. 954) objects attached to it, this operation will fail with **dds::core::PreconditionNotMetError** (p. 1941).

If there are still existing discovered **dds::sub::DataReader** (p. 976) s with the same `filter_name` and the filter's compile method of the filter have previously been called on the discovered **dds::sub::DataReader** (p. 976) s, finalize method of the filter will be called on those discovered **dds::sub::DataReader** (p. 976) s before the content filter is unregistered. This means filtering will now be performed on the application that is creating the **dds::sub::DataReader** (p. 976).

Parameters

filter_name	Name of the filter to unregister.

Exceptions

One	of the standard **Exceptions** (p. 427), or **dds::core::PreconditionNotMetError** (p. 1941)

8.86.4.33 unregister_type()

```
void unregister_type (
            const std::string & name )
```

<<***extension***>> *(p. 354)* Unregister a type that has previously been registered to this **dds::domain::Domain↩ Participant** (p. 1326).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Parameters

name	The name of the type to unregister

8.86.4.34 is_type_registered()

```
bool is_type_registered (
            const std::string & name ) const
```

<<***extension***>> *(*p. *354)* Check if a type has previously been registered to this **dds::domain::DomainParticipant** (p. 1326).

Parameters

name	The name of the type to check

Returns

bool true if it is registered, false otherwise

8.86.4.35 add_peer()

```
void add_peer (
            const std::string & peer_descr_string )
```

<<***extension***>> *(*p. *354)* Attempt to contact one or more additional peer participants.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Add the given peer description to the list of peers with which this **dds::domain::DomainParticipant** (p. 1326) will try to communicate.

This method may be called at any time after this **dds::domain::DomainParticipant** (p. 1326) has been created (before or after it has been enabled).

If this method is called after **dds::core::Entity::enable** (p. 1524), an attempt will be made to contact the new peer(s) immediately.

If this method is called *before* the **DomainParticipant** (p. 1326) is enabled, the peer description will simply be added to the list that was populated by **rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial↩_peers)** (p. 1268); the first attempted contact will take place after this **dds::domain::DomainParticipant** (p. 1326) is enabled.

Any peers added with the add_peer operation are equivalent to entries in the initial_peers list. Therefore, the participant will only discover remote participants using the newly added peer descriptor if a remote participant exists at that location and is configured to accept new peers (i.e., **rti::core::policy::Discovery::accept_unknown_peers** (p. 1270) is set to true). If multiple remote participants exist at that location (for example, if add_peer is called with a multicast address), the participant may discover multiple remote participants.

To be informed of the exact remote participants that are discovered, regardless of which peers this **dds::domain::DomainParticipant** (p. 1326) *attempts* to discover, use the built-in participant topic: **dds::topic::participant_topic_name()** (p. 441).

To remove specific peer locators, you may use **dds::domain::DomainParticipant::remove_peer(const std::string & peer_descr_string)** (p. 1353). If a peer is removed, the add_peer operation will add it back to the list of peers.

To stop communicating with a peer **dds::domain::DomainParticipant** (p. 1326) that has been discovered, use **dds::domain::ignore** (p. 635).

Adding a peer description with this method has no effect on the **rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers)** (p. 1268) that may be subsequently retrieved with **dds::domain::DomainParticipant::qos() const** (p. 1336)() (because **rti::core::policy::Discovery** (p. 1265) is immutable).

Parameters

peer_descr_string	New peer descriptor to be added. The format is specified in **Peer Descriptor Format** (p. 560).

Exceptions

One	of the standard **Exceptions** (p. 427)

8.86.4.36 remove_peer()

```
void remove_peer (
            const std::string & peer_descr_string )
```

*<<**extension**>> (p. 354)* Remove one or more peer participants from the list of peers with which this **dds::domain::DomainParticipant** (p. 1326) will try to communicate.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This method may be called any time after this **dds::domain::DomainParticipant** (p. 1326) has been enabled

Calling this method has the following effects:

- If a **dds::domain::DomainParticipant** (p. 1326) was already discovered, it will be locally removed along with all its entities.
- Any further requests coming from a **dds::domain::DomainParticipant** (p. 1326) located on any of the removed peers will be ignored.
- All the locators contained in the peer description will be removed from the peer list. The local **dds::domain::DomainParticipant** (p. 1326) will stop sending announcement to those locators.

If remote participants located on a peer that was previously removed are discovered, they will be ignored untill the related peer is added back by using **dds::domain::DomainParticipant::add_peer(const std::string & peer_descr_string)** (p. 1352).

Removing a peer description with this method has no effect on the **rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers)** (p. 1268) that may be subsequently retrieved with **dds::domain::DomainParticipant::qos() const** (p. 1336)() (because **rti::core::policy::Discovery** (p. 1265) is immutable).

Parameters

peer_descr_string	Peer descriptor to be removed. The format is specified in **Peer Descriptor Format** (p. 560).

Exceptions

One	of the standard **Exceptions** (p. 427)

8.86.4.37 dns_tracker_polling_period() [1/2]

```
dds::core::Duration dns_tracker_polling_period ( ) const
```

Retrieves the frequency used by the DNS tracker thread to query the DNS service.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The DNS tracker queries the DNS for hostnames specified in the initial peers of a **DomainParticipant** (p. 1326). The frequency of these queries is defined by **rti::core::policy::DiscoveryConfig::dns_tracker_polling_period** (p. 1311). If the value returned is **dds::core::Duration::infinite()** (p. 1451), the DNS tracker is disabled.

Returns

The DNS polling period

Exceptions

One	of the standard **Exceptions** (p. 427)

See also

rti::core::policy::DiscoveryConfig::dns_tracker_polling_period (p. 1311)

8.86.4.38 dns_tracker_polling_period() [2/2]

```
void dns_tracker_polling_period (
            const  dds::core::Duration & polling_period )
```

Configures the frequency in which the DNS tracker queries the DNS service.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This API allows you to change the frequency of the polling period for the DNS tracker. The range of accepted values, in seconds, goes from 1 second to 1 year. **dds::core::Duration::infinite()** (p. 1451) is also accepted as a valid value. If the duration is set to **dds::core::Duration::infinite()** (p. 1451), the DNS tracker is disabled.

Modifying the DNS tracker polling period through this has no effect on the **rti::core::policy::DiscoveryConfig::dns_↩ tracker_polling_period** (p. 1311) when it is retrieved with **dds::domain::DomainParticipant::qos() const** (p. 1336)().

Parameters

polling_period	Polling period to be configured.

Exceptions

One	of the standard **Exceptions** (p. 427)

See also

rti::core::policy::DiscoveryConfig::dns_tracker_polling_period (p. 1311)

dds::domain::DomainParticipant::add_peer(const std::string & peer_descr_string) (p. 1352)

8.86.4.39 resume_endpoint_discovery()

```
void resume_endpoint_discovery (
            const  dds::core::InstanceHandle & remote_participant_handle )
```

<<***extension***>> *(p. 354)* Initiates endpoint discovery with the remote **dds::domain::DomainParticipant** (p. 1326) identified by its InstanceHandle.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

If the operation completes successfully, the **dds::domain::DomainParticipant** (p. 1326) will initiate endpoint discovery with the remote **dds::domain::DomainParticipant** (p. 1326) provided as a parameter.

When **rti::core::policy::Discovery::enable_endpoint_discovery** (p. 1271) is set to false, this operation allows the RTI Connext application to select for which remote DomainParticipants endpoint discovery is performed. By disabling endpoint discovery, the **DomainParticipant** (p. 1326) will not store any state about remote endpoints and will not send local endpoint information to remote DomainParticipants.

If **rti::core::policy::Discovery::enable_endpoint_discovery** (p. 1271) is set to true, endpoint discovery will automatically occur for every discovered **dds::domain::DomainParticipant** (p. 1326). In this case, invoking this operation will have no effect and will return successfully.

When **rti::core::policy::Discovery::enable_endpoint_discovery** (p. 1271) is set to false, you have two options after a remote **dds::domain::DomainParticipant** (p. 1326) is discovered:

- Call this operation to enable endpoint discovery. After invoking this operation, the **dds::domain::DomainParticipant** (p. 1326) will start to exchange endpoint information so that matching and communication can occur with the remote **dds::domain::DomainParticipant** (p. 1326).
- Call the **dds::domain::ignore** (p. 635) operation to permanently ignore endpoint discovery with the remote **dds::domain::DomainParticipant** (p. 1326).

Setting **rti::core::policy::Discovery::enable_endpoint_discovery** (p. 1271) to false enables application-level authentication use cases, in which a **dds::domain::DomainParticipant** (p. 1326) will initiate endpoint discovery with a remote **dds::domain::DomainParticipant** (p. 1326) after successful authentication at the application level.

The `remote_participant_handle` paremeter is the one that appears in the **dds::sub::SampleInfo** (p. 2285) retrieved when reading the data samples available for the built-in ParticipantBuiltinTopicDataDataReader.

If the specified remote **dds::domain::DomainParticipant** (p. 1326) is not in the database of discovered DomainParticipants or has been previously ignored, this operation will fail with **dds::core::Error** (p. 1539).

This operation can be called multiple times on the same remote participant. If endpoint discovery has already been resumed, successive calls will have no effect and will return successfully.

Parameters

remote_participant_handle	<<***in***>> *(p. 355)* Handle of a discovered **dds::domain::DomainParticipant** (p. 1326) for which endpoint discovery is to be resumed.

Exceptions

One	of the **Standard Exceptions** (p. 428),or **dds::core::NotEnabledError** (p. 1869)

See also

rti::core::policy::Discovery (p. 1265)

8.86.4.40 delete_durable_subscription()

```
void delete_durable_subscription (
            const  rti::core::EndpointGroup & group )
```

<<***extension***>> *(p. 354)* Deletes an existing Durable Subscription on all Persistence Services.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The Persistence Service will delete the Durable Subscription and the quorum of the existing samples will be considered satisfied.

Parameters

group	<<***in***>> *(p. 355)* **rti::core::EndpointGroup** (p. 1515) specifying the Durable Subscription name. Quorum is not required for this operation.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.86.4.41 register_durable_subscription()

```
void register_durable_subscription (
            const  rti::core::EndpointGroup & group,
            const std::string & topic_name )
```

<<***extension***>> *(p. 354)* Registers a Durable Subscription on the specified **dds::topic::Topic** (p. 2474) on all Persistence Services.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

If you need to receive all samples published on a **dds::topic::Topic** (p. 2474), including the ones published while a **dds::sub::DataReader** (p. 976) is inactive or before it may be created, create a Durable Subscription using this method.

In this way, the Persistence Service will ensure that all the samples on that **dds::topic::Topic** (p. 2474) are retained until they are acknowledged by at least *N* DataReaders belonging to the Durable Subscription where *N* is the quorum count.

If the same Durable Subscription is created on a different **dds::topic::Topic** (p. 2474), the Persistence Service will implicitly delete the previous Durable Subscription and create a new one on the new **dds::topic::Topic** (p. 2474).

Parameters

group	<<***in***>> *(p. 355)* **rti::core::EndpointGroup** (p. 1515) The Durable Subscription name and quorum.
topic_name	<<***in***>> *(p. 355)* The topic name for which the Durable Subscription is created.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.86.4.42 participant_protocol_status()

```
rti::core::status::DomainParticipantProtocolStatus participant_protocol_status ( )
```

<<***extension***>> *(p. 354)* Get the protocol status for this participant

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This also resets the status so that it is no longer considered changed.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869).

8.86.4.43 time_until_license_expiration()

```
rti::core::int64 time_until_license_expiration ( ) const
```

<<***extension***>> *(p. 354)* Retrieve the remaining time until the license validated for this **DomainParticipant** (p. 1326) expires, with seconds precision.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This operation reports, with seconds precision, the remaining time until the license validated for this **dds::domain::↩ DomainParticipant** (p. 1326) expires. If the license does not expire, the reported value is the maximum value representable by a signed 64-bit integer . If the license is already expired, the reported value is zero.

Note: This operation is only available in license-managed RTI Connext distributions.

Exceptions

One	of the **Standard Exceptions** (p. 428)

Returns

Remaining time until license expiration, in seconds.

8.86.5 Friends And Related Function Documentation

8.86.5.1 ignore() [1/2]

```
void ignore (
            const  dds::domain::DomainParticipant & participant,
            const  dds::core::InstanceHandle & handle )  [related]
```

Instructs RTI Connext to locally ignore a remote **dds::domain::DomainParticipant** (p. 1326).

```
#include <dds/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **dds::domain** (p. 634)

From the time of this call onwards, RTI Connext will locally behave as if the remote participant did not exist. This means it will ignore any topic, publication, or subscription that originates on that **dds::domain::DomainParticipant** (p. 1326).

There is no way to reverse this operation.

This operation can be used in conjunction with the discovery of remote participants offered by means of the **dds::topic::ParticipantBuiltinTopicData** (p. 1908) to provide access control.

Application data can be associated with a **dds::domain::DomainParticipant** (p. 1326) by means of the **USER_DATA** (p. 554) policy. This application data is propagated as a field in the built-in topic and can be used by an application to implement its own access control policy.

The **dds::domain::DomainParticipant** (p. 1326) to ignore is identified by the handle argument. This `handle` is the one that appears in the **dds::sub::SampleInfo** (p. 2285) retrieved when reading the data-samples available for the built-in **dds::sub::DataReader** (p. 976) to the **dds::domain::DomainParticipant** (p. 1326) topic. The built-in **dds::sub::DataReader** (p. 976) is read with the same **dds::sub::DataReader::read** (p. 996) and **dds::sub::DataReader::take** (p. 997) operations used for any **dds::sub::DataReader** (p. 976).

Parameters

participant	The **DomainParticipant** (p. 1326) for which the remote entity will be ignored.
handle	The **dds::core::InstanceHandle** (p. 1612) of the remote entity that will be ignored.

8.86.5.2 ignore() [2/2]

```
template<typename FwdIterator >
void ignore (
            const  dds::domain::DomainParticipant & participant,
            FwdIterator begin,
            FwdIterator end )  [related]
```

Instructs RTI Connext to locally ignore a group of remote **dds::domain::DomainParticipant** (p. 1326).

```
#include <dds/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **dds::domain** (p. 634)

The series of entities whose instance handles are made available via the provided iterators will be ignored.

Parameters

participant	The **DomainParticipant** (p. 1326) for which the remote entity will be ignored
begin	The begin iterator for the series of InstanceHandles to ignore
end	The end iterator for the series of InstanceHandles to ignore

See also

ignore(const dds::domain::DomainParticipant& participant, const dds::core::InstanceHandle& handle) (p. 1359)

References **dds::domain::ignore()**.

8.86.5.3 discovered_participants() [1/2]

```
dds::core::InstanceHandleSeq discovered_participants (
            const  dds::domain::DomainParticipant & participant )  [related]
```

Retrieves the list of other participants discovered by this participant.

```
#include <dds/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **dds::domain** (p. 634)

This operation retrieves the list of **dds::domain::DomainParticipant** (p. 1326) entities that have been discovered in the domain and that the application has not indicated should be "ignored" by means of the **dds::domain::ignore** (p. 635) operation. When using **rti::core::policy::DiscoveryConfigBuiltinPluginKindMask::SPDP2** (p. 1324), this list only includes **dds::domain::DomainParticipant** (p. 1326) entities that the application has received configuration information from.

Parameters

participant	The **DomainParticipant** (p. 1326) where to look up the discovered participants

Returns

The list of `InstanceHandles` that can be passed to **dds::domain::discovered_participant_data** (p. 638).

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869)

8.86.5.4 discovered_participants() [2/2]

```
template<typename FwdIterator >
FwdIterator discovered_participants (
            const  dds::domain::DomainParticipant & participant,
            FwdIterator begin,
            FwdIterator end )  [related]
```

Retrieves the list of other participants discovered by this participant.

```
#include <dds/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **dds::domain** (p. 634)

Template Parameters

FwdIterator	A forward iterator whose value type is **dds::core::InstanceHandleSeq** (p. 619)

This overload copies the instance handles into an iterator range.

Parameters

participant	The participant whose discovered participants are looked up
begin	The begin iterator where to copy the instance handles
end	One past the last position where instance handles are copied

References **dds::domain::discovered_participants()**.

8.86.5.5 discovered_participant_data()

```
dds::topic::ParticipantBuiltinTopicData discovered_participant_data (
            const  dds::domain::DomainParticipant & participant,
            const  dds::core::InstanceHandle & participant_handle )  [related]
```

Retrieves the information about one participant discovered by this participant.

```
#include <dds/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **dds::domain** (p. 634)

This operation retrieves information on a **dds::domain::DomainParticipant** (p. 1326) that has been discovered on the network. The participant must be in the same domain as the participant on which this operation is invoked and must not have been "ignored" by means of the **dds::domain::ignore** (p. 635) operation.

The `participant_handle` must correspond to such a **DomainParticipant** (p. 1326). Otherwise, the operation will fail with **dds::core::PreconditionNotMetError** (p. 1941).

Use the operation **dds::domain::discovered_participants** (p. 637) to find the **dds::domain::DomainParticipant** (p. 1326) entities that are currently discovered.

MT Safety:

Safe.

Parameters

participant	The **DomainParticipant** (p. 1326) where to look up the information
participant_handle	<<***in***>> *(p. 355)* **dds::core::InstanceHandle** (p. 1612) of **dds::domain::DomainParticipant** (p. 1326).

Returns

The participant information about `participant_handle`

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::PreconditionNotMetError** (p. 1941) or **dds::core::NotEnabledError** (p. 1869)

See also

dds::topic::ParticipantBuiltinTopicData (p. 1908)

dds::domain::discovered_participants (p. 637)

8.86.5.6 find()

```
DomainParticipant find (
            int32_t domain_id )  [related]
```

Locates an existing **dds::domain::DomainParticipant** (p. 1326).

```
#include <dds/domain/find.hpp>
```

Note

This is a standalone function in the namespace **dds::domain** (p. 634)

If no such **DomainParticipant** (p. 1326) exists, the operation will return **dds::core::null** (p. 438).

If multiple DomainParticipants belonging to that domain id exist, then the operation will return one of them. It is not specified which one.

Parameters

domain_id	The domain id of the **DomainParticipant** (p. 1326) to find

8.86.5.7 banish_ignored_participants()

```
void banish_ignored_participants (
            const  dds::domain::DomainParticipant & participant )  [related]
```

<<***extension***>> *(p. 354)* Prevents ignored remote DomainParticipants from receiving traffic from the local **DomainParticipant** (p. 1326).

```
#include <rti/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **rti::domain** (p. 727)

This method complements **dds::domain::ignore** (p. 635): ignore_participant prevents the local **dds::domain::DomainParticipant** (p. 1326) from processing traffic from the remote **DomainParticipant** (p. 1326), while this method prevents already ignored remote DomainParticipants from processing traffic from the local **DomainParticipant** (p. 1326).

Note: this method is currently only supported when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

MT Safety:

Safe.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::PreconditionNotMetError** (p. 1941), **dds::core::NotEnabledError** (p. 1869)

See also

dds::domain::ignore (p. 635)

8.86.5.8 discovered_participant_subject_name()

```
rti::core::optional_value< std::string > discovered_participant_subject_name (
            const  dds::domain::DomainParticipant & participant,
            const  dds::core::InstanceHandle & participant_handle )  [related]
```

<<***extension***>> *(p. 354)* Returns **rti::core::policy::EntityName::name** (p. 1531) for the specified **Domain↩ Participant** (p. 1326).

```
#include <rti/domain/discovery.hpp>
```

Note

This is a standalone function in the namespace **rti::domain** (p. 727)

This operation retrieves the **rti::core::policy::EntityName::name** (p. 1531) of a **dds::domain::DomainParticipant** (p. 1326) that has been discovered on the network. The participant must be in the same domain as the participant on which this operation is invoked and must not have been "ignored" by means of the **dds::domain::ignore** (p. 635) operation.

The `participant_handle` must correspond to such a **DomainParticipant** (p. 1326). If the `participant↩ _handle` is **dds::core::InstanceHandle::nil()** (p. 1614) or is not a valid **dds::core::InstanceHandle** (p. 1612) for a **DomainParticipant** (p. 1326), then the operation will fail with **dds::core::InvalidArgumentError** (p. 1619). If the `participant_handle` corresponds to a **DomainParticipant** (p. 1326) that has not been discovered, then the operation will fail with **dds::core::PreconditionNotMetError** (p. 1941).

Use the operation **dds::domain::discovered_participants** (p. 637) to find the **dds::domain::DomainParticipant** (p. 1326) entities that are currently discovered.

Note: this method has different functionality when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

MT Safety:

Safe.

Parameters

participant	The **DomainParticipant** (p. 1326) that has discovered the discovered participant.
participant_handle	<<***in***>> *(p. 355)* **dds::core::InstanceHandle** (p. 1612) of **dds::domain::DomainParticipant** (p. 1326).

Returns

The **rti::core::policy::EntityName::name** (p. 1531) for the `participant_handle`

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::PreconditionNotMetError** (p. 1941) or **dds::core::NotEnabledError** (p. 1869)

See also

rti::core::policy::EntityName::name (p. 1531)

dds::domain::discovered_participants (p. 637)

8.86.5.9 discovered_participants_from_subject_name()

```
dds::core::InstanceHandleSeq discovered_participants_from_subject_name (
            const  dds::domain::DomainParticipant & participant,
            const  rti::core::optional_value< std::string > & subject_name )  [related]
```

<<***extension***>> *(p. 354)* Returns a list of discovered **DomainParticipant** (p. 1326) entities that have the given **rti::core::policy::EntityName::name** (p. 1531).

`#include <rti/domain/discovery.hpp>`

Note

This is a standalone function in the namespace **rti::domain** (p. 727)

This operation retrieves the same list as **dds::domain::discovered_participants** (p. 637), except this list contains only the participants that have the given **rti::core::policy::EntityName::name** (p. 1531).

Note: this method has different functionality when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

MT Safety:

Safe.

Parameters

participant	The **DomainParticipant** (p. 1326) whose discovered participants this operation will look up.
subject_name	<<***in***>> *(p. 355)* The **rti::core::policy::EntityName::name** (p. 1531) by which to filter the list.

Returns

The list of `InstanceHandles` corresponding to participants with the given **rti::core::policy::EntityName↩ ::name** (p. 1531).

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869)

8.86.5.10 find_participant_by_name()

```
dds::domain::DomainParticipant find_participant_by_name (
            const std::string & participant_name )  [related]
```

<<***extension***>> *(p. 354)* Locates an existing **dds::domain::DomainParticipant** (p. 1326) by name.

Note

`#include <rti/domain/find.hpp>` This function is in the **`rti::domain`** (p. 727) namespace

If no such **DomainParticipant** (p. 1326) exists, the operation will return **dds::core::null** (p. 438).

8.86.5.11 find_participants() [1/2]

```
template<typename ParticipantFwdIterator >
uint32_t find_participants (
            ParticipantFwdIterator begin,
            uint32_t max_size )  [related]
```

<<***extension***>> *(p. 354)* Retrieves all the participants created by the application up to a maximum number.

Note

`#include <rti/domain/find.hpp>` This function is in the **`rti::domain`** (p. 727) namespace

If no such **DomainParticipant** (p. 1326) exists, the operation will return **dds::core::null** (p. 438).

DomainParticipants are ordered in the output sequence from the most recently created to the oldest.

MT Safety:

Unsafe. Creating or deleting DomainParticipants at the same time as calling this operation is not safe.

Template Parameters

ParticipantFwdIterator	A forward iterator whose value type is **dds::domain::DomainParticipant** (p. 1326)

Parameters

begin	The iterator where to begin adding DomainParticipants.
max_size	The maximum number of elements to add

Returns

The number of elements added

8.86.5.12 find_participants() [2/2]

```
template<typename ParticipantFwdIterator >
uint32_t find_participants (
             ParticipantFwdIterator begin )  [related]
```

<<***extension***>> *(p. 354)* Retrieves all the participants created by the application.

Note

`#include <rti/domain/find.hpp>` This function is in the **`rti::domain`** (p. 727) namespace

If no such **DomainParticipant** (p. 1326) exists, the operation will return **dds::core::null** (p. 438).

DomainParticipants are ordered in the output sequence from the most recently created to the oldest.

MT Safety:

Unsafe. Creating or deleting DomainParticipants at the same time as calling this operation is not safe.

Template Parameters

ParticipantFwdIterator	A forward iterator whose value type is **dds::domain::DomainParticipant** (p. 1326)

Parameters

begin	The iterator where to begin adding DomainParticipants.

Returns

The number of elements added

8.86.5.13 find_type()

```
const  dds::core::xtypes::DynamicType & find_type (
             const  dds::domain::DomainParticipant & participant,
             const std::string & type_name )  [related]
```

<<***extension***>> *(p. 354)* Retrieves a type registered with this participant

`#include <rti/domain/find.hpp>`

Note

This is a standalone function in the namespace **rti::domain** (p. 727)

Every data type used in a **DomainParticipant** (p. 1326) has a registered type name, which in most cases is the same as the type's name (the name used to define the type), but it can be different.

Types are registered by the creation of a **dds::topic::Topic** (p. 2474), whose constructors optionally receive a registered type name different from the type's name.

Types can also be registered when parsing an XML configuration and calling **dds::core::QosProvider::create_participant_from_config()** (p. 2046).

Parameters

participant	The **DomainParticipant** (p. 1326) where the type is registered
type_name	The name used to register the type in this participant

Returns

If `type_name` exists in this `participant`, this function returns the DynamicType describing the type. It can be cast down to **dds::core::xtypes::StructType** (p. 2400) or **dds::core::xtypes::UnionType** (p. 2584), depending on whether the type is a struct or a union.

Exceptions

dds::core::Error *(p. 1539)*	If the type doesn't exist or the operation fails for any other reason

Example:

```
dds::domain::DomainParticipant participant(0);
dds::topic::Topic<Foo> topic1(participant, "Foo Topic"); // registered as "Foo"
dds::topic::Topic<Foo> topic2(participant, "MyFoo Topic", "MyFoo"); // registered as "MyFoo"
```

```
const auto& type1 = rti::domain::find_type(participant, "Foo");
const auto& type2 = rti::domain::find_type(participant, "MyFoo");
std::cout « type1 « std::endl;
std::cout « type2 « std::endl;
assert(type1 == type2); // Same type registered with different names
// Cast to StructType to access more information about the type
const auto& type1_struct =
        static_cast<const dds::core::xtypes::StructType&>(type1);
std::cout « type1_struct.member(0).name() « std::endl;
```

8.86.5.14 register_type() [1/2]

```
void register_type (
              dds::domain::DomainParticipant & participant,
             const std::string & name,
             const  dds::core::xtypes::DynamicType & type,
             const  rti::core::xtypes::DynamicDataTypeSerializationProperty & serialization_↩
property =  rti::core::xtypes::DynamicDataTypeSerializationProperty::DEFAULT )  [related]
```

<<***extension***>> *(*p. *354)* Registers a DynamicType with specific serialization properties

Typically you don't need to call this function, since the topic constructor takes care of that automatically. You do need to call this function before creating the topic if you want to change the default data-serialization property.

Calling this function also allows to change the registered name of the type, which by default is `type.name()`.

Parameters

participant	The participant where to register this type
name	The name to use to register this type
type	The type definition
serialization_property	The data-serialization property

See also

dds::core::xtypes::DynamicType (p. 1504)

dds::core::xtypes::DynamicData (p. 1461)

Examples

Foo.hpp.

8.86.5.15 register_type() [2/2]

```
template<typename T >
void register_type (
```

```
            const std::string & registered_type_name =  dds::topic::topic_type_name<T>::value()
) [related]
```

<<***extension***>> *(p. 354)* Registers a User-Generated Type with RTI Connext. This function is used along with XML Application Creation.

When using XML Application creation, you must use this function to register any user-generated types with RTI Connext before creating your system.

When you don't use XML Application creation, you don't typically need to call this function, since the topic constructor takes care of that automatically.

Template Parameters

T	The user-generated type that is being registered

Parameters

registered_type_name	The name to use when registering the type. This is the name that will be used in your XML configuration file to refer to the type.

See also

dds::domain::DomainParticipant::create_participant_from_config

XML Application Creation (p. 342)

8.87 rti::domain::DomainParticipantConfigParams Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Input paramaters for creating a participant from xml configuration. It allows modification of some of the properties of the entities defined in the configuration.

```
#include "rti/domain/DomainParticipantConfigParams.hpp"
```

Public Member Functions

- **DomainParticipantConfigParams** (int32_t the_domain_id= **DOMAIN_ID_USE_XML_CONFIG**, const std↩ ::string &the_participant_name= **ENTITY_NAME_USE_XML_CONFIG**, const std::string &the_participant_qos↩ _library_name= **QOS_ELEMENT_NAME_USE_XML_CONFIG**, const std::string &the_participant_qos_profile↩ _name= **QOS_ELEMENT_NAME_USE_XML_CONFIG**, const std::string &the_domain_entity_qos_library_↩ name= **QOS_ELEMENT_NAME_USE_XML_CONFIG**, const std::string &the_domain_entity_qos_profile_name= **QOS_ELEMENT_NAME_USE_XML_CONFIG**)

  *Create a **DomainParticipantConfigParams** (p. 1371) object with the specified values.*
- int32_t **domain_id** () const

  Get the domain id.
- **DomainParticipantConfigParams** & **domain_id** (int32_t the_domain_id)

Set the domain id from which the DomainParticipant is created.

- std::string **participant_name** () const

  Get the participant name.

- **DomainParticipantConfigParams** & **participant_name** (const std::string &the_participant_name)

  Set the name assigned to the DomainParticipant.

- std::string **participant_qos_library_name** () const

  Get the participant qos library name.

- **DomainParticipantConfigParams** & **participant_qos_library_name** (const std::string &the_participant_qos↩ _library_name)

  Set the QoS library name containing the QoS profile from which the DDS_DomainParticipant is created.

- std::string **participant_qos_profile_name** () const

  Get the participant qos profile name.

- **DomainParticipantConfigParams** & **participant_qos_profile_name** (const std::string &the_participant_qos↩ _profile_name)

  Set the QoS profile name from which the DomainParticipant is created.

- std::string **domain_entity_qos_library_name** () const

  Get the domain entity qos library name.

- **DomainParticipantConfigParams** & **domain_entity_qos_library_name** (const std::string &the_domain_↩ entity_qos_library_name)

  Set the QoS library name containing the QoS profile from which the all the entities defined under the participant configuration are created.

- std::string **domain_entity_qos_profile_name** () const

  Get the domain entity qos profile name.

- **DomainParticipantConfigParams** & **domain_entity_qos_profile_name** (const std::string &the_domain_↩ entity_qos_profile_name)

  Set the QoS profile name from which the all the entities defined under the participant configuration are created.

Static Public Attributes

- static OMG_DDS_API_CLASS_VARIABLE const std::string **ENTITY_NAME_USE_XML_CONFIG**

  Special value to be used to indicate that a participant should be created with an autogenerated entity name.

- static OMG_DDS_API_CLASS_VARIABLE const std::string **QOS_ELEMENT_NAME_USE_XML_CONFIG**

  Special value to be used to indicate that entities should be created from the QoS profile specified in the participant configuration.

- static OMG_DDS_API_CLASS_VARIABLE const int32_t **DOMAIN_ID_USE_XML_CONFIG**

  Special value to be used to indicate that a participant should be created using the domain ID specified in the participant configuration.

8.87.1 Detailed Description

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Input paramaters for creating a participant from xml configuration. It allows modification of some of the properties of the entities defined in the configuration.

See also

dds::core::QosProvider::create_participant_from_config (p. 2046)

8.87.2 Constructor & Destructor Documentation

8.87.2.1 DomainParticipantConfigParams()

```
rti::domain::DomainParticipantConfigParams::DomainParticipantConfigParams (
            int32_t the_domain_id =  DOMAIN_ID_USE_XML_CONFIG,
            const std::string & the_participant_name =  ENTITY_NAME_USE_XML_CONFIG,
            const std::string & the_participant_qos_library_name =  QOS_ELEMENT_NAME_USE_XML_↩
CONFIG,
            const std::string & the_participant_qos_profile_name =  QOS_ELEMENT_NAME_USE_XML_↩
CONFIG,
            const std::string & the_domain_entity_qos_library_name =  QOS_ELEMENT_NAME_USE_XML_↩
CONFIG,
            const std::string & the_domain_entity_qos_profile_name =  QOS_ELEMENT_NAME_USE_XML_↩
CONFIG )  [inline]
```

Create a **DomainParticipantConfigParams** (p. 1371) object with the specified values.

All parameters are optional. If ommitted, the value for that parameter will be retrieved from the xml configuration.

Parameters

the_domain_id	Domain ID from which the DomainParticipant is created.
the_participant_name	The name assigned to the DomainParticipant
the_participant_qos_library_name	QoS library name containing the QoS profile from which the DDS_DomainParticipant is created.
the_participant_qos_profile_name	QoS profile name from which the DomainParticipant is created.
the_domain_entity_qos_library_name	QoS library name containing the QoS profile from which the all the entities defined under the participant configuration are created.
the_domain_entity_qos_profile_name	QoS profile name from which the all the entities defined under the participant configuration are created.

8.87.3 Member Function Documentation

8.87.3.1 domain_id() [1/2]

```
int32_t rti::domain::DomainParticipantConfigParams::domain_id ( ) const  [inline]
```

Get the domain id.

See also

domain_id(int32_t) (p. 1373)

8.87.3.2 domain_id() [2/2]

```
DomainParticipantConfigParams & rti::domain::DomainParticipantConfigParams::domain_id (
            int32_t the_domain_id )  [inline]
```

Set the domain id from which the DomainParticipant is created.

Allows overriding the domain ID defined in the configuration for the participant to be created. If the special value DOMAIN_ID_USE_XML_CONFIG is specified then the ID in the configuration will be used.

8.87.3.3 participant_name() [1/2]

```
std::string rti::domain::DomainParticipantConfigParams::participant_name ( ) const  [inline]
```

Get the participant name.

See also

participant_name(const std::string&) (p. 1374)

8.87.3.4 participant_name() [2/2]

```
DomainParticipantConfigParams & rti::domain::DomainParticipantConfigParams::participant_name (
            const std::string & the_participant_name )  [inline]
```

Set the name assigned to the DomainParticipant.

This is the name the name that will be set in the dds::domain::qos::DomainParticipantQos::participant_name. It allows overriding the participant name that is generated automatically.

When this member is lexicographically equal to the special value ENTITY_NAME_USE_XML_CONFIG then an automatically generated name will be assigned.

8.87.3.5 participant_qos_library_name() [1/2]

```
std::string rti::domain::DomainParticipantConfigParams::participant_qos_library_name ( ) const
[inline]
```

Get the participant qos library name.

See also

participant_qos_library_name(const std::string&) (p. 1374)

8.87.3.6 participant_qos_library_name() [2/2]

```
DomainParticipantConfigParams & rti::domain::DomainParticipantConfigParams::participant_qos_↩
library_name (
            const std::string & the_participant_qos_library_name )  [inline]
```

Set the QoS library name containing the QoS profile from which the DDS_DomainParticipant is created.

Allows overriding the QoS defined in the configuration for the participant to be created. This value only affects to the **dds::domain::DomainParticipant** (p. 1326).

When this member is lexicographically equal to the special value QOS_ELEMENT_NAME_USE_XML_CONFIG then the QoS library from the configuration will be applied. Also, the same action will apply for the QoS profile and the value in **rti::domain::DomainParticipantConfigParams::participant_qos_profile_name** (p. 1375) will be ignored.

8.87.3.7 participant_qos_profile_name() [1/2]

```
std::string rti::domain::DomainParticipantConfigParams::participant_qos_profile_name ( ) const
[inline]
```

Get the participant qos profile name.

See also

participant_qos_profile_name(const std::string&) (p. 1375)

8.87.3.8 participant_qos_profile_name() [2/2]

```
DomainParticipantConfigParams & rti::domain::DomainParticipantConfigParams::participant_qos_↩
profile_name (
            const std::string & the_participant_qos_profile_name )  [inline]
```

Set the QoS profile name from which the DomainParticipant is created.

Allows overriding the QoS defined in the configuration for the participant to be created.This value only affects to the **dds::domain::DomainParticipant** (p. 1326).

When this member is lexicographically equal to the special value QOS_ELEMENT_NAME_USE_XML_CONFIG then the QoS profile from the configuration will be applied. Also, the same action will apply for the QoS library and the value in **rti::domain::DomainParticipantConfigParams::participant_qos_library_name** (p. 1374) will be ignored.

8.87.3.9 domain_entity_qos_library_name() [1/2]

```
std::string rti::domain::DomainParticipantConfigParams::domain_entity_qos_library_name ( ) const
[inline]
```

Get the domain entity qos library name.

See also

domain_entity_qos_library_name(const std::string&) (p. 1376)

8.87.3.10 domain_entity_qos_library_name() [2/2]

```
DomainParticipantConfigParams & rti::domain::DomainParticipantConfigParams::domain_entity_qos_↩
library_name (
            const std::string & the_domain_entity_qos_library_name )  [inline]
```

Set the QoS library name containing the QoS profile from which the all the entities defined under the participant configuration are created.

Allows overriding the QoS defined in the configuration for the domain entities to be created. This value only affects to the DomainEntity.

When this member is lexicographically equal to the special value QOS_ELEMENT_NAME_USE_XML_CONFIG then the QoS library from the configuration will be applied. Also, the same action will apply for the QoS profile and the value in **rti::domain::DomainParticipantConfigParams::domain_entity_qos_profile_name** (p. 1376) will be ignored.

8.87.3.11 domain_entity_qos_profile_name() [1/2]

```
std::string rti::domain::DomainParticipantConfigParams::domain_entity_qos_profile_name ( ) const
[inline]
```

Get the domain entity qos profile name.

See also

DomainParticipantConfigParams (p. 1371)& **domain_entity_qos_profile_name(const std::string&)** (p. 1376)

8.87.3.12 domain_entity_qos_profile_name() [2/2]

```
DomainParticipantConfigParams & rti::domain::DomainParticipantConfigParams::domain_entity_qos_↩
profile_name (
            const std::string & the_domain_entity_qos_profile_name )  [inline]
```

Set the QoS profile name from which the all the entities defined under the participant configuration are created.

Allows overriding the QoS defined in the configuration for the domain entities to be created. This value only affects to the DomainEntity.

When this member is lexicographically equal to the special value QOS_ELEMENT_NAME_USE_XML_CONFIG then the QoS profile from the configuration will be applied. Also, the same action will apply for the QoS library and the value in **rti::domain::DomainParticipantConfigParams::domain_entity_qos_library_name** (p. 1375) will be ignored.

8.87.4 Member Data Documentation

8.87.4.1 ENTITY_NAME_USE_XML_CONFIG

```
OMG_DDS_API_CLASS_VARIABLE const std::string rti::domain::DomainParticipantConfigParams::ENTITY_↩
NAME_USE_XML_CONFIG  [static]
```

Special value to be used to indicate that a participant should be created with an autogenerated entity name.

8.87.4.2 QOS_ELEMENT_NAME_USE_XML_CONFIG

```
OMG_DDS_API_CLASS_VARIABLE const std::string rti::domain::DomainParticipantConfigParams::QOS_↩
ELEMENT_NAME_USE_XML_CONFIG  [static]
```

Special value to be used to indicate that entities should be created from the QoS profile specified in the participant configuration.

8.87.4.3 DOMAIN_ID_USE_XML_CONFIG

```
OMG_DDS_API_CLASS_VARIABLE const int32_t rti::domain::DomainParticipantConfigParams::DOMAIN_ID_↩
USE_XML_CONFIG  [static]
```

Special value to be used to indicate that a participant should be created using the domain ID specified in the participant configuration.

8.88 dds::domain::qos::DomainParticipantFactoryQos Class Reference

<<***value-type***>> *(p. 350)* Container of the QoS policies that do not apply to a specific entity

```
#include "dds/domain/qos/DomainParticipantFactoryQos.hpp"
```

Public Member Functions

- template<typename POLICY >
  const POLICY & **policy** () const

  Gets a QoS policy by const reference.
- template<typename POLICY >
  POLICY & **policy** ()

  Gets a QoS policy by reference.

Related Functions

(Note that these are not member functions.)

- std::string **to_string** (const **DomainParticipantFactoryQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::domain::qos::DomainParticipantFactoryQos*** *(p. 1378)*
- std::string **to_string** (const **DomainParticipantFactoryQos** &qos, const **DomainParticipantFactoryQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::domain::qos::DomainParticipantFactoryQos*** *(p. 1378)*
- std::string **to_string** (const **DomainParticipantFactoryQos** &qos, const **rti::core::qos_print_all_t** &qos_↩ print_all, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::domain::qos::DomainParticipantFactoryQos*** *(p. 1378)*
- std::ostream & **operator**<< (std::ostream &out, const rti::domain::qos::DomainParticipantFactoryQos &qos)

  <<***extension***>> *(p. 354) Prints a dds::sub::qos::DomainParticipantFactoryQos to an output stream.*

8.88.1 Detailed Description

<<***value-type***>> *(p. 350)* Container of the QoS policies that do not apply to a specific entity

To set or get this policies, use **DomainParticipant::participant_factory_qos()** (p. 1340).

See also

Qos Use Cases (p. 602)

8.88.2 Member Function Documentation

8.88.2.1 policy() [1/2]

```
template<typename POLICY >
const POLICY & dds::domain::qos::DomainParticipantFactoryQos::policy ( ) const
```

Gets a QoS policy by const reference.

See also

policy() (p. 1379)

8.88.2.2 policy() [2/2]

```
template<typename POLICY >
POLICY & dds::domain::qos::DomainParticipantFactoryQos::policy ( )
```

Gets a QoS policy by reference.

Template Parameters

Policy	One of the DomainParticipantFactory QoS policies: • **dds::core::policy::EntityFactory** (p. 1527), • **rti::core::policy::SystemResourceLimits** (p. 2443)

Note

In other language APIs the **DomainParticipantFactoryQos** (p. 1378) also contains the policies to configure the loading of Qos profiles in XML (ProfileQosPolicy) and the logging configuration. These are configured through **dds::core::QosProvider** (p. 2025) and **rti::config::Logger** (p. 1683) respectively.

See also

DDSQosModule_set_qos

8.88.3 Friends And Related Function Documentation

8.88.3.1 to_string() [1/3]

```
std::string to_string (
            const  DomainParticipantFactoryQos & qos,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
DomainParticipantFactoryQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for DomainParticipantFactoryQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
DomainParticipantFactoryQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

8.88.3.2 to_string() [2/3]

```
std::string to_string (
            const  DomainParticipantFactoryQos & qos,
            const  DomainParticipantFactoryQos & base,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

8.88.3.3 to_string() [3/3]

```
std::string to_string (
            const  DomainParticipantFactoryQos & qos,
            const  rti::core::qos_print_all_t & qos_print_all,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378)

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
```
to_string(qos, rti::core::qos_print_all);
```

Returns

The string representation of the qos

8.88.3.4 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const rti::domain::qos::DomainParticipantFactoryQos & qos )  [related]
```

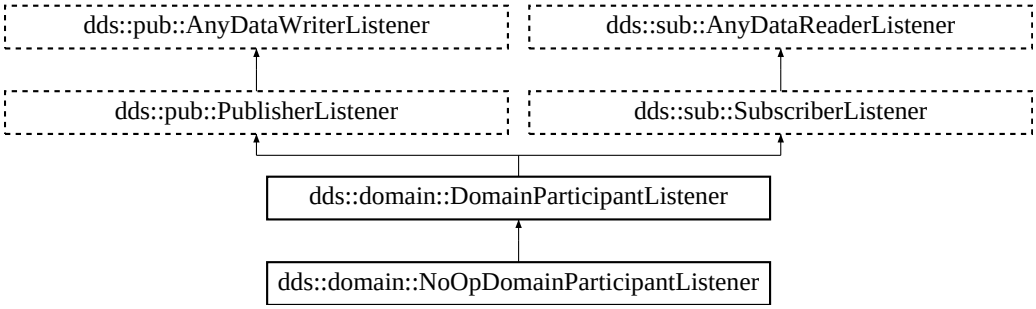
<<***extension***>> *(p. 354)* Prints a dds::sub::qos::DomainParticipantFactoryQos to an output stream.

8.89 dds::domain::DomainParticipantListener Class Reference

The listener class for a **DomainParticipant** (p. 1326).

```
#include "dds/domain/DomainParticipantListener.hpp"
```

Inheritance diagram for dds::domain::DomainParticipantListener:



Public Member Functions

- virtual void **on_invalid_local_status_advance_notice** (**DomainParticipant** &participant, const **rti::core**↩ **::status::InvalidLocalAdvanceNoticeStatus** &status)=0

  <<***extension***>> *(p. 354) Notifies that the identity of the local* ***DomainParticipant*** *(p. 1326) is about to expire.*

8.89.1 Detailed Description

The listener class for a **DomainParticipant** (p. 1326).

8.89.2 Member Function Documentation

8.89.2.1 on_invalid_local_status_advance_notice()

```
virtual void dds::domain::DomainParticipantListener::on_invalid_local_status_advance_notice (
             DomainParticipant & participant,
            const  rti::core::status::InvalidLocalAdvanceNoticeStatus & status )  [pure virtual]
```

<<***extension***>> *(*p. *354)* Notifies that the identity of the local **DomainParticipant** (p. 1326) is about to expire.

Implemented in **dds::domain::NoOpDomainParticipantListener** (p. 1849).

8.90 rti::core::status::DomainParticipantProtocolStatus Class Reference

```
#include <Status.hpp>
```

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- int64_t **corrupted_rtps_message_count** () const

  The number of corrupted RTPS messages detected by the domain participant.
- int64_t **corrupted_rtps_message_count_change** () const

  The incremental change in the number of corrupted RTPS messages detected by the domain participant since the last time the status was read.
- **dds::core::Time last_corrupted_message_timestamp** () const

  The timestamp when the last corrupted RTPS message was detected by the domain participant.

8.90.1 Detailed Description

Entity:

dds::domain::DomainParticipant (p. 1326) The corrupted messages are detected by validating the received CRC. The participant protocol status can be obtained by enabling **rti::core::policy::WireProtocol::compute_crc** (p. 2640) and **rti::core::policy::WireProtocol::check_crc** (p. 2640) at the publishing and subscribing application respectively.

8.90.2 Member Function Documentation

8.90.2.1 corrupted_rtps_message_count()

```
int64_t rti::core::status::DomainParticipantProtocolStatus::corrupted_rtps_message_count ( ) const
[inline]
```

The number of corrupted RTPS messages detected by the domain participant.

Counts the corrupted RTPS messages received by the participant. It includes messages belonging to discovery and user traffic.

8.90.2.2 corrupted_rtps_message_count_change()

```
int64_t rti::core::status::DomainParticipantProtocolStatus::corrupted_rtps_message_count_change (
) const  [inline]
```

The incremental change in the number of corrupted RTPS messages detected by the domain participant since the last time the status was read.

8.90.2.3 last_corrupted_message_timestamp()

```
 dds::core::Time rti::core::status::DomainParticipantProtocolStatus::last_corrupted_message_↩
timestamp ( ) const  [inline]
```

The timestamp when the last corrupted RTPS message was detected by the domain participant.

8.91 dds::domain::qos::DomainParticipantQos Class Reference

<<***value-type***>> *(*p. *350)* Container of the QoS policies that a **dds::domain::DomainParticipant** (p. 1326) supports

```
#include <dds/domain/qos/DomainParticipantQos.hpp>
```

Public Member Functions

- **DomainParticipantQos** ()

  *Creates a **DomainParticipantQos** (*p. *1384) with the default value for each policy.*
- template<typename POLICY >
  const POLICY & **policy** () const

  Gets a QoS policy by const reference.
- template<typename POLICY >
  POLICY & **policy** ()

  Gets a QoS policy by reference.
- template<typename Policy >
  DomainParticipantQos & **policy** (const Policy &p)

  Sets a policy.
- template<typename Policy >
  DomainParticipantQos & **operator**<< (const Policy &p)

  Sets a policy.
- template<typename Policy >
  const **DomainParticipantQos** & **operator**>> (Policy &p) const

  Copies the values of a policy.

Related Functions

(Note that these are not member functions.)

- std::string **to_string** (const **DomainParticipantQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::domain::qos::DomainParticipantQos*** *(p. 1384)*
- std::string **to_string** (const **DomainParticipantQos** &qos, const **DomainParticipantQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::domain::qos::DomainParticipantQos*** *(p. 1384)*
- std::string **to_string** (const **DomainParticipantQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::domain::qos::DomainParticipantQos*** *(p. 1384)*
- std::ostream & **operator<<** (std::ostream &out, const rti::domain::qos::DomainParticipantQos &qos)

  <<***extension***>> *(p. 354) Prints a dds::sub::qos::DomainParticipantQos to an output stream.*

8.91.1 Detailed Description

<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::domain::DomainParticipant** (p. 1326) supports

8.91.2 DomainParticipantQos Policies

A **DomainParticipantQos** (p. 1384) contains the following policies:

- **dds::core::policy::UserData** (p. 2591),
- **dds::core::policy::EntityFactory** (p. 1527),
- **rti::core::policy::WireProtocol** (p. 2630),
- **rti::core::policy::TransportBuiltin** (p. 2533),
- **rti::core::policy::TransportUnicast** (p. 2556),
- **rti::core::policy::Discovery** (p. 1265),
- **rti::core::policy::DomainParticipantResourceLimits** (p. 1391),
- **rti::core::policy::Event** (p. 1541),
- **rti::core::policy::ReceiverPool** (p. 2144),
- **rti::core::policy::Database** (p. 969),
- **rti::core::policy::DiscoveryConfig** (p. 1271),
- **rti::core::policy::Property** (p. 1969),
- **rti::core::policy::EntityName** (p. 1530),
- **rti::core::policy::TransportMulticastMapping** (p. 2547)

- **rti::core::policy::Service** (p. 2350)
- rti::core::policy::Partition

To get or set policies use the **policy()** (p. 1387) getters and setters or operator << (see **examples** (p. 603)).

Certain members must be set in a consistent manner:

Length of dds::domain::qos::DomainParticipantQos::user_data .value <= dds::domain::qos::DomainParticipantQos↩ ::resource_limits .participant_user_data_max_length

For dds::domain::qos::DomainParticipantQos::discovery_config .publication_writer high_watermark <= dds::domain::qos::DomainParticipantQos::resource_limits .local_writer_allocation .max_count heartbeats_per_max_samples <= dds::domain::qos::DomainParticipantQos::resource_limits .local_writer_allocation.↩ max_count

For dds::domain::qos::DomainParticipantQos::discovery_config .suscription_writer high_watermark <= dds::domain::qos::DomainParticipantQos::resource_limits .local_reader_allocation.max_↩ count heartbeats_per_max_samples <= dds::domain::qos::DomainParticipantQos::resource_limits .local_reader↩ _allocation.max_count

If any of the above are not true, **dds::domain::DomainParticipant::qos(const dds::domain::qos::Domain↩ ParticipantQos&)** (p. 1337) and dds::domain::DomainParticipant::set_qos_with_profile and **dds::domain::Domain↩ Participant::default_participant_qos()** (p. 1341) will fail with **dds::core::InconsistentPolicyError** (p. 1610), and the **dds::domain::DomainParticipant** (p. 1326) constructors will fail with **dds::core::Error** (p. 1539)

Entity:

dds::domain::DomainParticipant (p. 1326)

See also

All QoS Policies (p. 508) and allowed ranges within each Qos.

NDDS_DISCOVERY_PEERS (p. 559)

Qos Use Cases (p. 602)

Examples

Foo_publisher.cxx, and **Foo_subscriber.cxx**.

8.91.3 Constructor & Destructor Documentation

8.91.3.1 DomainParticipantQos()

```
dds::domain::qos::DomainParticipantQos::DomainParticipantQos ( )
```

Creates a **DomainParticipantQos** (p. 1384) with the default value for each policy.

8.91.4 Member Function Documentation

8.91.4.1 policy() [1/3]

```
template<typename POLICY >
const POLICY & dds::domain::qos::DomainParticipantQos::policy ( ) const
```

Gets a QoS policy by const reference.

Template Parameters

Policy	One of the **DomainParticipantQos Policies** (p. 1385)

See also

Setting Qos Values (p. 603)

8.91.4.2 policy() [2/3]

```
template<typename POLICY >
POLICY & dds::domain::qos::DomainParticipantQos::policy ( )
```

Gets a QoS policy by reference.

Template Parameters

Policy	One of the **DomainParticipantQos Policies** (p. 1385)

See also

Setting Qos Values (p. 603)

8.91.4.3 policy() [3/3]

```
template<typename Policy >
DomainParticipantQos & dds::domain::qos::DomainParticipantQos::policy (
            const Policy & p )  [inline]
```

Sets a policy.

See also

policy() (p. 1387)

Setting Qos Values (p. 603)

8.91.4.4 operator<<()

```
template<typename Policy >
DomainParticipantQos & dds::domain::qos::DomainParticipantQos::operator<< (
            const Policy & p )  [inline]
```

Sets a policy.

See also

policy() (p. 1387)

Setting Qos Values (p. 603)

8.91.4.5 operator>>()

```
template<typename Policy >
const  DomainParticipantQos & dds::domain::qos::DomainParticipantQos::operator>> (
            Policy & p ) const  [inline]
```

Copies the values of a policy.

Parameters

p	The destination where to copy the current value of the Policy

See also

policy() (p. 1387)

Setting Qos Values (p. 603)

8.91.5 Friends And Related Function Documentation

8.91.5.1 to_string() [1/3]

```
std::string to_string (
            const  DomainParticipantQos & qos,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::domain::qos::DomainParticipantQos** (p. 1384)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
DomainParticipantQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for DomainParticipantQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
DomainParticipantQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

8.91.5.2 to_string() [2/3]

```
std::string to_string (
            const  DomainParticipantQos & qos,
            const  DomainParticipantQos & base,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::domain::qos::DomainParticipantQos** (p. 1384)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

8.91.5.3 to_string() [3/3]

```
std::string to_string (
            const  DomainParticipantQos & qos,
            const  rti::core::qos_print_all_t & qos_print_all,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(*p. *354)* Obtains a string representation of the **dds::domain::qos::DomainParticipantQos** (p. 1384)

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
```
to_string(qos, rti::core::qos_print_all);
```

Returns

The string representation of the qos

8.91.5.4 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const rti::domain::qos::DomainParticipantQos & qos )  [related]
```

<<***extension***>> *(*p. *354)* Prints a dds::sub::qos::DomainParticipantQos to an output stream.

8.92 rti::core::policy::DomainParticipantResourceLimits Class Reference

<<***extension***>> *(p. 116)* Configures the memory usage of certain **dds::domain::DomainParticipant** (p. 1326) resources

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **DomainParticipantResourceLimits** & **local_writer_allocation** (const **AllocationSettings** &settings)

  <<***extension***>> *(p. 116) Allocation settings applied to local DataWriters.*
- **AllocationSettings** **local_writer_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **local_reader_allocation** (const **AllocationSettings** &settings)

  <<***extension***>> *(p. 116) Allocation settings applied to local DataReaders.*
- **AllocationSettings** **local_reader_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **local_publisher_allocation** (const **AllocationSettings** &settings)

  <<***extension***>> *(p. 116) Allocation settings applied to local Publisher.*
- **AllocationSettings** **local_publisher_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **local_subscriber_allocation** (const **AllocationSettings** &settings)

  <<***extension***>> *(p. 116) Allocation settings applied to local Subscriber.*
- **DomainParticipantResourceLimits** & **local_topic_allocation** (const **AllocationSettings** &settings)

  <<***extension***>> *(p. 116) Allocation settings applied to local Topic.*
- **AllocationSettings** **local_topic_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **remote_writer_allocation** (const **AllocationSettings** &settings)

  <<***extension***>> *(p. 116) Allocation settings applied to remote DataWriters.*
- **AllocationSettings** **remote_writer_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **remote_reader_allocation** (const **AllocationSettings** &settings)

  <<***extension***>> *(p. 116) Allocation settings applied to remote DataReaders.*
- **AllocationSettings** **remote_reader_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **remote_participant_allocation** (const **AllocationSettings** &settings)

  <<***extension***>> *(p. 116) Allocation settings applied to remote DomainParticipants.*
- **AllocationSettings** **remote_participant_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **matching_writer_reader_pair_allocation** (const **AllocationSettings** &settings)

  <<***extension***>> *(p. 116) Allocation settings applied to matching local writer and remote/local reader pairs.*
- **AllocationSettings** **matching_writer_reader_pair_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **matching_reader_writer_pair_allocation** (const **AllocationSettings** &settings)

  <<***extension***>> *(p. 116) Allocation settings applied to matching local reader and remote/local writer pairs.*

- **AllocationSettings matching_reader_writer_pair_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **ignored_entity_allocation** (const **AllocationSettings** &settings)

  *<<**extension**>> (p. 116) Allocation settings applied to ignored entities.*
- **AllocationSettings ignored_entity_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **content_filtered_topic_allocation** (const **AllocationSettings** &settings)

  *<<**extension**>> (p. 116) Allocation settings applied to content filtered topic.*
- **AllocationSettings content_filtered_topic_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **content_filter_allocation** (const **AllocationSettings** &settings)

  *<<**extension**>> (p. 116) Allocation settings applied to content filter.*
- **AllocationSettings content_filter_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **read_condition_allocation** (const **AllocationSettings** &settings)

  *<<**extension**>> (p. 116) Allocation settings applied to read condition pool.*
- **AllocationSettings read_condition_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **query_condition_allocation** (const **AllocationSettings** &settings)

  *<<**extension**>> (p. 116) Allocation settings applied to query condition pool.*
- **AllocationSettings query_condition_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **outstanding_asynchronous_sample_allocation** (const **Allocation↩ Settings** &settings)

  *<<**extension**>> (p. 116) Allocation settings applied to the maximum number of samples (from all **dds::pub::DataWriter** (p. 1138)) waiting to be asynchronously written.*
- **AllocationSettings outstanding_asynchronous_sample_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **flow_controller_allocation** (const **AllocationSettings** &settings)

  *<<**extension**>> (p. 116) Allocation settings applied to flow controllers.*
- **AllocationSettings flow_controller_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **local_writer_hash_buckets** (int32_t hash_buckets)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Hash_Buckets settings applied to local DataWriters.*
- int32_t **local_writer_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **local_reader_hash_buckets** (int32_t hash_buckets)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Number of hash buckets for local DataReaders.*
- int32_t **local_reader_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **local_publisher_hash_buckets** (int32_t hash_buckets)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Number of hash buckets for local Publisher.*
- int32_t **local_publisher_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **local_subscriber_hash_buckets** (int32_t hash_buckets)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Number of hash buckets for local Subscriber.*

- int32_t **local_subscriber_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **local_topic_hash_buckets** (int32_t hash_buckets)

  [DEPRECATED] *(*p. ***296)*** *<<**extension**>> (*p. *116) Number of hash buckets for local Topic.*
- int32_t **local_topic_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **remote_writer_hash_buckets** (int32_t hash_buckets)

  [DEPRECATED] *(*p. ***296)*** *<<**extension**>> (*p. *116) Number of hash buckets for remote DataWriters.*
- int32_t **remote_writer_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **remote_reader_hash_buckets** (int32_t hash_buckets)

  [DEPRECATED] *(*p. ***296)*** *<<**extension**>> (*p. *116) Number of hash buckets for remote DataReaders.*
- int32_t **remote_reader_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **remote_participant_hash_buckets** (int32_t hash_buckets)

  [DEPRECATED] *(*p. ***296)*** *<<**extension**>> (*p. *116) Number of hash buckets for remote DomainParticipants.*
- int32_t **remote_participant_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **matching_writer_reader_pair_hash_buckets** (int32_t hash_buckets)

  [DEPRECATED] *(*p. ***296)*** *<<**extension**>> (*p. *116) Number of hash buckets for matching local writer and remote/local reader pairs.*
- int32_t **matching_writer_reader_pair_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **matching_reader_writer_pair_hash_buckets** (int32_t hash_buckets)

  [DEPRECATED] *(*p. ***296)*** *<<**extension**>> (*p. *116) Number of hash buckets for matching local reader and remote/local writer pairs.*
- int32_t **matching_reader_writer_pair_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **ignored_entity_hash_buckets** (int32_t hash_buckets)

  [DEPRECATED] *(*p. ***296)*** *<<**extension**>> (*p. *116) Number of hash buckets for ignored entities.*
- int32_t **ignored_entity_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **content_filtered_topic_hash_buckets** (int32_t hash_buckets)

  [DEPRECATED] *(*p. ***296)*** *<<**extension**>> (*p. *116) Number of hash buckets for content filtered topics.*
- int32_t **content_filtered_topic_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **content_filter_hash_buckets** (int32_t hash_buckets)

  [DEPRECATED] *(*p. ***296)*** *<<**extension**>> (*p. *116) Number of hash buckets for content filters.*
- int32_t **content_filter_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **flow_controller_hash_buckets** (int32_t hash_buckets)

  [DEPRECATED] *(*p. ***296)*** *<<**extension**>> (*p. *116) Number of hash buckets for flow controllers.*
- int32_t **flow_controller_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **max_gather_destinations** (int32_t max_destinations)

  *<<**extension**>> (*p. *116) Maximum number of destinations per RTI Connext send.*
- int32_t **max_gather_destinations** () const

Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **participant_user_data_max_length** (int32_t max_length)

  *<<**extension**>> (p. 116) Maximum length of user data in **dds::domain::qos::DomainParticipantQos** (p. 1384) and **dds::topic::ParticipantBuiltinTopicData** (p. 1908)*
- int32_t **participant_user_data_max_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **topic_data_max_length** (int32_t max_length)

  *<<**extension**>> (p. 116) Maximum length of topic data in **dds::topic::qos::TopicQos** (p. 2509), **dds::topic::TopicBuiltinTopicData** (p. 2493), **dds::topic::PublicationBuiltinTopicData** (p. 1977) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)*
- int32_t **topic_data_max_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **publisher_group_data_max_length** (int32_t max_length)

  *<<**extension**>> (p. 116) Maximum length of group data in **dds::pub::qos::PublisherQos** (p. 2007) and **dds::topic::PublicationBuiltinTopicData** (p. 1977)*
- int32_t **publisher_group_data_max_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **subscriber_group_data_max_length** (int32_t max_length)

  *<<**extension**>> (p. 116) Maximum length of group data in **dds::sub::qos::SubscriberQos** (p. 2423) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)*
- int32_t **subscriber_group_data_max_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **writer_user_data_max_length** (int32_t max_length)

  *<<**extension**>> (p. 116) Maximum length of user data in **dds::pub::qos::DataWriterQos** (p. 1226) and **dds::topic::PublicationBuiltinTopicData** (p. 1977)*
- int32_t **writer_user_data_max_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **reader_user_data_max_length** (int32_t max_length)

  *<<**extension**>> (p. 116) Maximum length of user data in **dds::sub::qos::DataReaderQos** (p. 1072) and **dds::topic::SubscriptionBuiltinTopicData** (p. 2428)*
- int32_t **reader_user_data_max_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **max_partitions** (int32_t partitions)

  *<<**extension**>> (p. 116) Maximum number of partition name strings allowable in a **dds::core::policy::Partition** (p. 1923).*
- int32_t **max_partitions** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **max_partition_cumulative_characters** (int32_t max_characters)

  *<<**extension**>> (p. 116) Maximum number of combined characters allowable in all partition names in a **dds::core::policy::Partition** (p. 1923).*
- **DomainParticipantResourceLimits** & **type_code_max_serialized_length** (int32_t max_length)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Maximum size of serialized string for type code.*
- int32_t **type_code_max_serialized_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **type_object_max_serialized_length** (int32_t max_length)

  *<<**extension**>> (p. 116) The maximum length, in bytes, that the buffer to serialize a TypeObject v1 can consume.*
- int32_t **type_object_max_serialized_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **type_object_max_deserialized_length** (int32_t max_length)

<<***extension***>> (p. *116) The maximum number of bytes that a deserialized TypeObject v1 can consume.*

- int32_t **type_object_max_deserialized_length** () const

  Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **deserialized_type_object_dynamic_allocation_threshold** (int32_t threshold)

  <<***extension***>> (p. *116) A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject v1.*

- int32_t **deserialized_type_object_dynamic_allocation_threshold** () const

  Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **serialized_type_object_dynamic_allocation_threshold** (int32_t threshold)

  <<***extension***>> (p. *116) A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject v1.*

- int32_t **serialized_type_object_dynamic_allocation_threshold** () const

  Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **contentfilter_property_max_length** (int32_t max_length)

  <<***extension***>> (p. *116) This field is the maximum length of all data related to a Content-filtered topic.*

- int32_t **contentfilter_property_max_length** () const

  Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **channel_seq_max_length** (int32_t max_length)

  <<***extension***>> (p. *116) Maximum number of channels that can be specified in **rti::core::policy::MultiChannel** (p. 1749) for **MultiChannel** (p. 1749) DataWriters.*

- int32_t **channel_seq_max_length** () const

  Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **channel_filter_expression_max_length** (int32_t max_length)

  <<***extension***>> (p. *116) Maximum length of a channel **rti::core::ChannelSettings::filter_expression** (p. 923) in a **MultiChannel** (p. 1749) DataWriter.*

- int32_t **channel_filter_expression_max_length** () const

  Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **participant_property_list_max_length** (int32_t max_length)

  <<***extension***>> (p. *116) Maximum number of properties associated with the **dds::domain::DomainParticipant** (p. 1326).*

- int32_t **participant_property_list_max_length** () const

  Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **participant_property_string_max_length** (int32_t max_length)

  <<***extension***>> (p. *116) Maximum string length of the properties associated with the **dds::domain::DomainParticipant** (p. 1326).*

- int32_t **participant_property_string_max_length** () const

  Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **writer_property_list_max_length** (int32_t max_length)

  <<***extension***>> (p. *116) Maximum number of properties associated with a **dds::pub::DataWriter** (p. 1138).*

- int32_t **writer_property_list_max_length** () const

  Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **writer_property_string_max_length** (int32_t max_length)

  <<***extension***>> (p. *116) Maximum string length of the properties associated with a **dds::pub::DataWriter** (p. 1138).*

- int32_t **writer_property_string_max_length** () const

  Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **reader_property_list_max_length** (int32_t max_length)

  <<***extension***>> (p. *116) Maximum number of properties associated with a **dds::sub::DataReader** (p. 976).*

- int32_t **reader_property_list_max_length** () const

Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **reader_property_string_max_length** (int32_t max_length)

  *<<**extension**>> (p. 116) Maximum string length of the properties associated with a **dds::sub::DataReader** (p. 976).*
- int32_t **reader_property_string_max_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **max_endpoint_groups** (int32_t max_groups)

  *<<**extension**>> (p. 116) Maximum number of **rti::core::EndpointGroup** (p. 1515) allowable in a **rti::core::policy::Availability** (p. 868).*
- int32_t **max_endpoint_groups** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **max_endpoint_group_cumulative_characters** (int32_t max_cumulative_characters)

  *<<**extension**>> (p. 116) Maximum number of combined role_name characters allowed in all **rti::core::EndpointGroup** (p. 1515) in a **rti::core::policy::Availability** (p. 868).*
- int32_t **max_endpoint_group_cumulative_characters** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **transport_info_list_max_length** (int32_t max_length)

  *<<**extension**>> (p. 116) Maximum number of installed transports to send and receive information about in **dds::topic::ParticipantBuiltinTopicData::transport_info** (p. 1914).*
- int32_t **transport_info_list_max_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **ignored_entity_replacement_kind** (**IgnoredEntityReplacementKind** max_length)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **rti::core::policy::DomainParticipantResourceLimits::ignored_entity_allocation** (p. 1403) are reached.*
- **IgnoredEntityReplacementKind** **ignored_entity_replacement_kind** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **remote_topic_query_allocation** (const **AllocationSettings** &settings)

  *<<**extension**>> (p. 116) Allocation settings applied to remote TopicQueries.*
- **AllocationSettings** **remote_topic_query_allocation** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **remote_topic_query_hash_buckets** (int32_t hash_buckets)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Number of hash buckets for remote TopicQueries.*
- int32_t **remote_topic_query_hash_buckets** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **writer_data_tag_list_max_length** (int32_t max_length)

  *<<**extension**>> (p. 116) Maximum number of data tags associated with a **dds::pub::DataWriter** (p. 1138).*
- int32_t **writer_data_tag_list_max_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **writer_data_tag_string_max_length** (int32_t max_length)

  *<<**extension**>> (p. 116) Maximum string length of the data tags associated with a **dds::pub::DataWriter** (p. 1138).*
- int32_t **writer_data_tag_string_max_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **reader_data_tag_list_max_length** (int32_t max_length)

  *<<**extension**>> (p. 116) Maximum number of data tags associated with a **dds::sub::DataReader** (p. 976).*
- int32_t **reader_data_tag_list_max_length** () const

  Getter (see setter with the same name)

- **DomainParticipantResourceLimits** & **reader_data_tag_string_max_length** (int32_t max_length)

  <<***extension***>> *(p. 116) Maximum string length of the data tags associated with a* ***dds::sub::DataReader*** *(p. 976).*
- int32_t **reader_data_tag_string_max_length** () const

  Getter (see setter with the same name)
- **DomainParticipantResourceLimits** & **shmem_ref_transfer_mode_max_segments** (uint32_t the_shmem↩ _ref_transfer_mode_max_segments)

  <<***extension***>> *(p. 116) Maximum number of segments created by all DataWriters belonging to a* ***dds::domain::↩ DomainParticipant*** *(p. 1326).*
- uint32_t **shmem_ref_transfer_mode_max_segments** () const

  Getter (see setter with the same name)

8.92.1 Detailed Description

<<***extension***>> *(p. 116)* Configures the memory usage of certain **dds::domain::DomainParticipant** (p. 1326) resources

This QoS policy sets maximum size limits on variable-length parameters used by the participant and its contained Entities. It also controls the initial and maximum sizes of data structures used by the participant to store information about locally-created and remotely-discovered entities (such as DataWriters/DataReaders), as well as parameters used by the internal database to size the hash tables it uses.

By default, a **dds::domain::DomainParticipant** (p. 1326) is allowed to dynamically allocate memory as needed as users create local Entities such as **dds::pub::DataWriter** (p. 1138) and **dds::sub::DataReader** (p. 976) or as the participant discovers new applications to store their information. By setting fixed values for the maximum parameters in this Qos↩ Policy, you can bound the memory that can be allocated by a DomainParticipant. In addition, by setting the initial values to the maximum values, you can reduce the amount of memory allocated by DomainParticipants after the initialization period. Notice that memory can still be allocated dynamically after the initialization period. For example, when a new local **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976) is created, the initial memory required for its queue is allocated dynamically.

The maximum sizes of different variable-length parameters such as the number of partitions that can be stored in the **dds::core::policy::Partition** (p. 1923), the maximum length of data store in the **dds::core::policy::UserData** (p. 2591) and **dds::core::policy::GroupData** (p. 1591), and many others can be changed from their defaults using this QoS policy. However, it is important that all DomainParticipants that need to communicate with each other use the *same set* of maximum values. Otherwise, when these parameters are propagated from one **dds::domain::DomainParticipant** (p. 1326) to another, a **dds::domain::DomainParticipant** (p. 1326) with a smaller maximum length may reject the parameter, resulting in an error.

An important parameter in this QoS policy that is often changed by users is **rti::core::policy::DomainParticipant↩ ResourceLimits::type_object_max_serialized_length** (p. 1419).

This QoS policy is an extension to the DDS standard.

Entity:

dds::domain::DomainParticipant (p. 1326)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.92.2 Member Function Documentation

8.92.2.1 local_writer_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::local_↩
writer_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to local DataWriters.

[default] initial_count = 16; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

References **rti::core::policy::auto_writer_depth()**.

8.92.2.2 local_writer_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::local_writer_allocation (
) const
```

Getter (see setter with the same name)

8.92.2.3 local_reader_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::local_↩
reader_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to local DataReaders.

[default] initial_count = 16; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.4 local_reader_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::local_reader_allocation (
) const
```

Getter (see setter with the same name)

8.92.2.5 local_publisher_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::local_↩
publisher_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to local Publisher.

[default] initial_count = 4; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.6 local_publisher_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::local_publisher_allocation
( ) const
```

Getter (see setter with the same name)

8.92.2.7 local_subscriber_allocation()

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::local_↩
subscriber_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to local Subscriber.

[default] initial_count = 4; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.8 local_topic_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::local_↩
topic_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to local Topic.

[default] initial_count = 16; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.9 local_topic_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::local_topic_allocation ( )
const
```

Getter (see setter with the same name)

8.92.2.10 remote_writer_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::remote_↩
writer_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to remote DataWriters.

Remote DataWriters include all DataWriters, both local and remote.

[default] initial_count = 64; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.11 remote_writer_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::remote_writer_allocation (
) const
```

Getter (see setter with the same name)

8.92.2.12 remote_reader_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::remote_↩
reader_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to remote DataReaders.

Remote DataReaders include all DataReaders, both local and remote.

[default] initial_count = 64; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.13 remote_reader_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::remote_reader_allocation (
) const
```

Getter (see setter with the same name)

8.92.2.14 remote_participant_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::remote_↩
participant_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to remote DomainParticipants.

Remote DomainParticipants include all DomainParticipants, both local and remote.

[default] initial_count = 16; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.15 remote_participant_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::remote_participant_↩
allocation ( ) const
```

Getter (see setter with the same name)

8.92.2.16 matching_writer_reader_pair_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::matching_↩
writer_reader_pair_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to matching local writer and remote/local reader pairs.

[default] initial_count = 32; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.17 matching_writer_reader_pair_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::matching_writer_reader_↩
pair_allocation ( ) const
```

Getter (see setter with the same name)

8.92.2.18 matching_reader_writer_pair_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::matching_↩
reader_writer_pair_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to matching local reader and remote/local writer pairs.

[default] initial_count = 32; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.19 matching_reader_writer_pair_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::matching_reader_writer_↩
pair_allocation ( ) const
```

Getter (see setter with the same name)

8.92.2.20 ignored_entity_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::ignored_↩
entity_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to ignored entities.

[default] initial_count = 8; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.21 ignored_entity_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::ignored_entity_allocation
( ) const
```

Getter (see setter with the same name)

8.92.2.22 content_filtered_topic_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::content_↩
filtered_topic_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to content filtered topic.

[default] initial_count = 4; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Content Filtering** (p. 165)

8.92.2.23 content_filtered_topic_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::content_filtered_topic_↩
allocation ( ) const
```

Getter (see setter with the same name)

8.92.2.24 content_filter_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::content_↩
filter_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to content filter.

[default] initial_count = 4; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Content Filtering** (p. 165)

8.92.2.25 content_filter_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::content_filter_allocation
( ) const
```

Getter (see setter with the same name)

8.92.2.26 read_condition_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::read_↩
condition_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to read condition pool.

[default] initial_count = 4; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438), incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.27 read_condition_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::read_condition_allocation
( ) const
```

Getter (see setter with the same name)

8.92.2.28 query_condition_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::query_↩
condition_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to query condition pool.

[default] initial_count = 4; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438), incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Content Filtering** (p. 165)

8.92.2.29 query_condition_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::query_condition_allocation
( ) const
```

Getter (see setter with the same name)

8.92.2.30 outstanding_asynchronous_sample_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::outstanding↩
_asynchronous_sample_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to the maximum number of samples (from all **dds::pub::Data↩ Writer** (p. 1138)) waiting to be asynchronously written.

[default] initial_count = 64; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438), incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Flow Control** (p. 188)

8.92.2.31 outstanding_asynchronous_sample_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::outstanding_asynchronous↩
_sample_allocation ( ) const
```

Getter (see setter with the same name)

8.92.2.32 flow_controller_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::flow_↩
controller_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(*p. *116)* Allocation settings applied to flow controllers.

[default] initial_count = 4; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438), incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Flow Control** (p. 188)

8.92.2.33 flow_controller_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::flow_controller_allocation
( ) const
```

Getter (see setter with the same name)

8.92.2.34 local_writer_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::local_↩
writer_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Hash_Buckets settings applied to local DataWriters.

[default] 4

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.35 local_writer_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::local_writer_hash_buckets ( ) const
```

Getter (see setter with the same name)

8.92.2.36 local_reader_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::local_↩
reader_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local DataReaders.

[default] 4

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.37 local_reader_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::local_reader_hash_buckets ( ) const
```

Getter (see setter with the same name)

8.92.2.38 local_publisher_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::local_↩
publisher_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local Publisher.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.39 local_publisher_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::local_publisher_hash_buckets ( ) const
```

Getter (see setter with the same name)

8.92.2.40 local_subscriber_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::local_↩
subscriber_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local Subscriber.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.41 local_subscriber_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::local_subscriber_hash_buckets ( )
const
```

Getter (see setter with the same name)

8.92.2.42 local_topic_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::local_↩
topic_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for local Topic.

[default] 4

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.43 local_topic_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::local_topic_hash_buckets ( ) const
```

Getter (see setter with the same name)

8.92.2.44 remote_writer_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::remote_writer_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote DataWriters.

Remote DataWriters include all DataWriters, both local and remote.

[default] 16

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.45 remote_writer_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::remote_writer_hash_buckets ( ) const
```

Getter (see setter with the same name)

8.92.2.46 remote_reader_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::remote_reader_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote DataReaders.

Remote DataReaders include all DataReaders, both local and remote.

[default] 16

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.47 remote_reader_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::remote_reader_hash_buckets ( ) const
```

Getter (see setter with the same name)

8.92.2.48 remote_participant_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::remote_↩
participant_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for remote DomainParticipants.

Remote DomainParticipants include all DomainParticipants, both local and remote.

[default] 4

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.49 remote_participant_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::remote_participant_hash_buckets ( )
const
```

Getter (see setter with the same name)

8.92.2.50 matching_writer_reader_pair_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::matching_↩
writer_reader_pair_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for matching local writer and remote/local reader pairs.

[default] 32

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.51 matching_writer_reader_pair_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::matching_writer_reader_pair_hash_↩
buckets ( ) const
```

Getter (see setter with the same name)

8.92.2.52 matching_reader_writer_pair_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::matching_↩
reader_writer_pair_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for matching local reader and remote/local writer pairs.

[default] 32

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.53 matching_reader_writer_pair_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::matching_reader_writer_pair_hash_↩
buckets ( ) const
```

Getter (see setter with the same name)

8.92.2.54 ignored_entity_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::ignored_↩
entity_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for ignored entities.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.55 ignored_entity_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::ignored_entity_hash_buckets ( ) const
```

Getter (see setter with the same name)

8.92.2.56 content_filtered_topic_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::content_↩
filtered_topic_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for content filtered topics.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Content Filtering** (p. 165)

8.92.2.57 content_filtered_topic_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::content_filtered_topic_hash_buckets (
) const
```

Getter (see setter with the same name)

8.92.2.58 content_filter_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::content_↩
filter_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for content filters.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Content Filtering** (p. 165)

8.92.2.59 content_filter_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::content_filter_hash_buckets ( ) const
```

Getter (see setter with the same name)

8.92.2.60 flow_controller_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::flow_↩
controller_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Number of hash buckets for flow controllers.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Flow Control** (p. 188)

8.92.2.61 flow_controller_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::flow_controller_hash_buckets ( ) const
```

Getter (see setter with the same name)

8.92.2.62 max_gather_destinations() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::max_↩
gather_destinations (
            int32_t max_destinations )
```

<<***extension***>> *(p. 116)* Maximum number of destinations per RTI Connext send.

When RTI Connext sends out a message, it has the capability to send to multiple destinations to be more efficient. The maximum number of destinations per RTI Connext send is specified by max_gather_destinations.

[default] 16

[range] [16, 1 million]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.63 max_gather_destinations() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::max_gather_destinations ( ) const
```

Getter (see setter with the same name)

8.92.2.64 participant_user_data_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::participant↩
_user_data_max_length (
            int32_t max_length )
```

<<***extension***>> *(p. 116)* Maximum length of user data in **dds::domain::qos::DomainParticipantQos** (p. 1384) and **dds::topic::ParticipantBuiltinTopicData** (p. 1908)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application-Defined Metadata** (p. 159)

8.92.2.65 participant_user_data_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::participant_user_data_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.66 topic_data_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::topic_↩
data_max_length (
            int32_t max_length )
```

<<***extension***>> *(p. 116)* Maximum length of topic data in **dds::topic::qos::TopicQos** (p. 2509), **dds::topic::Topic↩ BuiltinTopicData** (p. 2493), **dds::topic::PublicationBuiltinTopicData** (p. 1977) and **dds::topic::Subscription↩ BuiltinTopicData** (p. 2428)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application-Defined Metadata** (p. 159)

8.92.2.67 topic_data_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::topic_data_max_length ( ) const
```

Getter (see setter with the same name)

8.92.2.68 publisher_group_data_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::publisher↩
_group_data_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum length of group data in **dds::pub::qos::PublisherQos** (p. 2007) and **dds::topic↩ ::PublicationBuiltinTopicData** (p. 1977)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application-Defined Metadata** (p. 159)

8.92.2.69 publisher_group_data_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::publisher_group_data_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.70 subscriber_group_data_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::subscriber↩
_group_data_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum length of group data in **dds::sub::qos::SubscriberQos** (p. 2423) and **dds↩ ::topic::SubscriptionBuiltinTopicData** (p. 2428)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application-Defined Metadata** (p. 159)

8.92.2.71 subscriber_group_data_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::subscriber_group_data_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.72 writer_user_data_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::writer_↩
user_data_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum length of user data in **dds::pub::qos::DataWriterQos** (p. 1226) and **dds::topic↩ ::PublicationBuiltinTopicData** (p. 1977)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application-Defined Metadata** (p. 159)

8.92.2.73 writer_user_data_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::writer_user_data_max_length ( ) const
```

Getter (see setter with the same name)

8.92.2.74 reader_user_data_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::reader_↩
user_data_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum length of user data in **dds::sub::qos::DataReaderQos** (p. 1072) and **dds↩ ::topic::SubscriptionBuiltinTopicData** (p. 2428)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application-Defined Metadata** (p. 159)

8.92.2.75 reader_user_data_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::reader_user_data_max_length ( ) const
```

Getter (see setter with the same name)

8.92.2.76 max_partitions() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::max_↩
partitions (
            int32_t partitions )
```

<<***extension***>> *(*p. *116)* Maximum number of partition name strings allowable in a **dds::core::policy::Partition** (p. 1923).

This setting is made on a per DomainParticipant basis; it cannot be set individually on a per Publisher/Subscriber basis. However, the limit is enforced and applies per Publisher/Subscriber.

This value cannot exceed 64.

[default] 64

[range] [0,64]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Discovery Isolation** (p. 183)

8.92.2.77 max_partitions() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::max_partitions ( ) const
```

Getter (see setter with the same name)

8.92.2.78 max_partition_cumulative_characters()

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::max_↩
partition_cumulative_characters (
            int32_t max_characters )
```

<<***extension***>> *(*p. *116)* Maximum number of combined characters allowable in all partition names in a **dds::core↩ ::policy::Partition** (p. 1923).

The maximum number of combined characters should account for a terminating NULL ('\0') character for each partition name string.

This setting is made on a per DomainParticipant basis; it cannot be set individually on a per Publisher/Subscriber basis. However, the limit is enforced and applies per Publisher/Subscriber.

This value cannot exceed 256.

[default] 256

[range] [0,256]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Discovery Isolation** (p. 183)

8.92.2.79 type_code_max_serialized_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::type_code↩
_max_serialized_length (
            int32_t max_length )
```

[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Maximum size of serialized string for type code.

This parameter is an alternative to **rti::core::policy::DomainParticipantResourceLimits::type_object_max_↩ serialized_length** (p. 1419) for limiting the size of the type code that a **dds::domain::DomainParticipant** (p. 1326) is able to store and propagate for user data types. Type codes can be used by external applications to understand user data types without having the data type predefined in compiled form. However, since type codes contain all of the information of a data structure, including the strings that define the names of the members of a structure, complex data structures can result in large type codes. So it is common for users to set this parameter to a large value, if used (by default, it is set to 0). However, as with all parameters in this QoS policy defining maximum sizes for variable-length elements, all DomainParticipants in the same domain should use the same value for this parameter.

Note: TypeObject v2 is now the standard, and default, method of exchanging type information in RTI Connext. It is recommended to keep **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::type_lookup_service** (p. 1321) in **rti::core::policy::DiscoveryConfig::enabled_builtin_channels** (p. 1294) enabled in DomainParticipants requiring type information.

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Type System** (p. 240)

8.92.2.80 type_code_max_serialized_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::type_code_max_serialized_length ( )
const
```

Getter (see setter with the same name)

8.92.2.81 type_object_max_serialized_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::type_↩
object_max_serialized_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* The maximum length, in bytes, that the buffer to serialize a TypeObject v1 can consume.

Type codes contain complete information about a data structure, including all member names; as a result, complex data structures can produce large type codes. This parameter limits the size of the resulting TypeObject v1 that a DomainParticipant is able to propagate.

If this parameter is set to a value between 1 and 0x7fffffff, the DomainParticipant will propagate TypeObject v1 for types that don't exceed the specified size. If **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::type_lookup↩ _service** (p. 1321) is also enabled, both TypeObject v1 and TypeObject v2 will be propagated (TypeObject v2s are not subject to this size limit).

If set to rti::core::LENGTH_AUTO, the DomainParticipant will not propagate TypeObject v1. If **rti::core::policy::↩ DiscoveryConfigBuiltinChannelKindMask::type_lookup_service** (p. 1321) is also enabled, only TypeObject v2 will be propagated.

If explicitly set to 0, the DomainParticipant won't propagate any TypeObject (neither TypeObject v1 nor TypeObject v2), regardless of whether **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::type_lookup_service** (p. 1321) is enabled.

Note: TypeObject v2 is now the standard, and default, method of exchanging type information in RTI Connext. It is recommended to enable **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::type_lookup_service** (p. 1321) in **rti::core::policy::DiscoveryConfig::enabled_builtin_channels** (p. 1294) in DomainParticipants requiring type validation.

[default] rti::core::LENGTH_AUTO

[range] [0,0x7fffffff], or rti::core::LENGTH_AUTO

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Type System** (p. 240)

8.92.2.82 type_object_max_serialized_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::type_object_max_serialized_length ( )
const
```

Getter (see setter with the same name)

8.92.2.83 type_object_max_deserialized_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::type_↩
object_max_deserialized_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* The maximum number of bytes that a deserialized TypeObject v1 can consume.

This parameter limits the size of the TypeObject v1 that a DomainParticipant is able to store.

Setting this field to 0 disables the ability of a DomainParticipant to receive TypeObject v1.

If **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::type_lookup_service** (p. 1321) is also enabled, the DomainParticipant will always send/receive TypeObject v2 (deserialized TypeObject v2s are not subject to this size limit), but will only receive TypeObject v1 if this parameter is set to a value greater than 0.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [0,0x7fffffff] or **dds::core::LENGTH_UNLIMITED** (p. 438)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Type System** (p. 240)

8.92.2.84 type_object_max_deserialized_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::type_object_max_deserialized_length (
) const
```

Getter (see setter with the same name)

8.92.2.85 deserialized_type_object_dynamic_allocation_threshold() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::deserialized↩
_type_object_dynamic_allocation_threshold (
            int32_t threshold )
```

<<***extension***>> *(p. 116)* A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject v1.

Above this threshold, the memory for a TypeObject v1 is allocated dynamically. Below it, the memory is obtained from a pool of fixed-size buffers. The size of the buffers is equal to this threshold.

In case **rti::core::policy::DomainParticipantResourceLimits::type_object_max_deserialized_length** (p. 1420) is different than **dds::core::LENGTH_UNLIMITED** (p. 438) and is smaller than **rti::core::policy::DomainParticipant↩ ResourceLimits::deserialized_type_object_dynamic_allocation_threshold** (p. 1420), **rti::core::policy::Domain↩ ParticipantResourceLimits::deserialized_type_object_dynamic_allocation_threshold** (p. 1420) will be adjusted to **rti::core::policy::DomainParticipantResourceLimits::type_object_max_deserialized_length** (p. 1420).

If **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::type_lookup_service** (p. 1321) is also enabled, this threshold does not affect memory allocation of TypeObject v2s.

[default] 4096

[range] [0,0x7fffffff] <= **rti::core::policy::DomainParticipantResourceLimits::type_object_max_deserialized_↩ length** (p. 1420)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Type System** (p. 240)

8.92.2.86 deserialized_type_object_dynamic_allocation_threshold() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::deserialized_type_object_dynamic_↩
allocation_threshold ( ) const
```

Getter (see setter with the same name)

8.92.2.87 serialized_type_object_dynamic_allocation_threshold() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::serialized↩
_type_object_dynamic_allocation_threshold (
            int32_t threshold )
```

<<***extension***>> *(p. 116)* A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject v1.

Above this threshold, the memory for a TypeObject v1 is allocated dynamically. Below it, the memory is obtained from a pool of fixed-size buffers. The size of the buffers is equal to this threshold.

The effective threshold value used by the middleware is determined as follows:

- If **rti::core::policy::DomainParticipantResourceLimits::type_object_max_serialized_length** (p. 1419) is rti↩ ::core::LENGTH_AUTO or 0, the effective threshold is 0 (no pre-allocated buffers).
- If this threshold is set to rti::core::LENGTH_AUTO and **rti::core::policy::DomainParticipantResourceLimits↩ ::type_object_max_serialized_length** (p. 1419) is set to a specific value, the effective threshold is min(type_↩ object_max_serialized_length, 8192).
- If the specified threshold exceeds type_object_max_serialized_length, it is automatically capped and adjusted down to match the maximum serialized length.

If **rti::core::policy::DiscoveryConfigBuiltinChannelKindMask::type_lookup_service** (p. 1321) is also enabled, this threshold does not affect memory allocation of TypeObject v2s.

[default] rti::core::LENGTH_AUTO

[range] [0,0x7fffffff], or rti::core::LENGTH_AUTO

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Type System** (p. 240)

8.92.2.88 serialized_type_object_dynamic_allocation_threshold() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::serialized_type_object_dynamic_↩
allocation_threshold ( ) const
```

Getter (see setter with the same name)

8.92.2.89 contentfilter_property_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::contentfilter↩
_property_max_length (
            int32_t max_length )
```

<<***extension***>> *(p. 116)* This field is the maximum length of all data related to a Content-filtered topic.

This is the sum of the length of the ContentFilteredTopic name, the length of the related topic name, the length of the filter expression, the length of the filter parameters, and the length of the filter name. The maximum number of combined characters should account for a terminating NULL ('\0') character for each string.

[default] 256

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Content Filtering** (p. 165)

8.92.2.90 contentfilter_property_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::contentfilter_property_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.91 channel_seq_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::channel_↩
seq_max_length (
            int32_t max_length )
```

<<***extension***>> *(p. 116)* Maximum number of channels that can be specified in **rti::core::policy::MultiChannel** (p. 1749) for **MultiChannel** (p. 1749) DataWriters.

[default] 32

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Content Filtering** (p. 165)

8.92.2.92 channel_seq_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::channel_seq_max_length ( ) const
```

Getter (see setter with the same name)

8.92.2.93 channel_filter_expression_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::channel_↩
filter_expression_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum length of a channel **rti::core::ChannelSettings::filter_expression** (p. 923) in a **MultiChannel** (p. 1749) DataWriter.

The length should account for a terminating NULL ('\0') character.

[default] 256

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Content Filtering** (p. 165)

8.92.2.94 channel_filter_expression_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::channel_filter_expression_max_length (
) const
```

Getter (see setter with the same name)

8.92.2.95 participant_property_list_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::participant↩
_property_list_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum number of properties associated with the **dds::domain::DomainParticipant** (p. 1326).

[default] 32

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application-Defined Metadata** (p. 159)

8.92.2.96 participant_property_list_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::participant_property_list_max_length (
) const
```

Getter (see setter with the same name)

8.92.2.97 participant_property_string_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::participant↩
_property_string_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum string length of the properties associated with the **dds::domain::Domain↩ Participant** (p. 1326).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with the **dds::domain::DomainParticipant** (p. 1326) properties.

[default] 4096

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application-Defined Metadata** (p. 159)

8.92.2.98 participant_property_string_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::participant_property_string_max_length
( ) const
```

Getter (see setter with the same name)

8.92.2.99 writer_property_list_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::writer_↩
property_list_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum number of properties associated with a **dds::pub::DataWriter** (p. 1138).

[range] [0,0xffff]

[default] 32

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application-Defined Metadata** (p. 159)

8.92.2.100 writer_property_list_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::writer_property_list_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.101 writer_property_string_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::writer_↩
property_string_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum string length of the properties associated with a **dds::pub::DataWriter** (p. 1138).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with the **dds::pub::DataWriter** (p. 1138) properties.

[default] 1024

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application-Defined Metadata** (p. 159)

8.92.2.102 writer_property_string_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::writer_property_string_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.103 reader_property_list_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::reader_↩
property_list_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum number of properties associated with a **dds::sub::DataReader** (p. 976).

[default] 32

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application-Defined Metadata** (p. 159)

8.92.2.104 reader_property_list_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::reader_property_list_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.105 reader_property_string_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::reader_↩
property_string_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum string length of the properties associated with a **dds::sub::DataReader** (p. 976).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with a **dds::sub::DataReader** (p. 976) properties.

[default] 1024

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Application-Defined Metadata** (p. 159)

8.92.2.106 reader_property_string_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::reader_property_string_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.107 max_endpoint_groups() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::max_↩
endpoint_groups (
            int32_t max_groups )
```

<<***extension***>> *(*p. *116)* Maximum number of **rti::core::EndpointGroup** (p. 1515) allowable in a **rti::core::policy↩ ::Availability** (p. 868).

[default] 32

[range] [0,65535]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.108 max_endpoint_groups() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::max_endpoint_groups ( ) const
```

Getter (see setter with the same name)

8.92.2.109 max_endpoint_group_cumulative_characters() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::max_↩
endpoint_group_cumulative_characters (
            int32_t max_cumulative_characters )
```

<<***extension***>> *(*p. *116)* Maximum number of combined role_name characters allowed in all **rti::core::Endpoint↩ Group** (p. 1515) in a **rti::core::policy::Availability** (p. 868).

The maximum number of combined characters should account for a terminating NULL character for each role_name string.

[default] 1024

[range] [0,65535]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.110 max_endpoint_group_cumulative_characters() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::max_endpoint_group_cumulative_characters
( ) const
```

Getter (see setter with the same name)

8.92.2.111 transport_info_list_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::transport↩
_info_list_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum number of installed transports to send and receive information about in **dds↩ ::topic::ParticipantBuiltinTopicData::transport_info** (p. 1914).

[default] 12

[range] [0,100]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Transports** (p. 237)

8.92.2.112 transport_info_list_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::transport_info_list_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.113 ignored_entity_replacement_kind() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::ignored_↩
entity_replacement_kind (
            IgnoredEntityReplacementKind max_length )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **rti::core::policy::DomainParticipantResourceLimits::ignored_entity↩ _allocation** (p. 1403) are reached.

When a **dds::domain::DomainParticipant** (p. 1326)'s number of ignored entities is greater than **rti::core::policy↩ ::DomainParticipantResourceLimits::ignored_entity_allocation** (p. 1403), the **dds::domain::DomainParticipant** (p. 1326) will try to make room by replacing an existing ignored participant entry. This field specifies what entity is allowed to be replaced.

If a replaceable participant entry is not available, an out-of-resources exception will be returned.

[default] DomainParticipantResourceLimitsIgnoredEntityReplacementKind::NO_REPLACEMENT_IGNORED_↩ ENTITY_REPLACEMENT

See also

DomainParticipantResourceLimitsIgnoredEntityReplacementKind

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.114 ignored_entity_replacement_kind() [2/2]

```
IgnoredEntityReplacementKind rti::core::policy::DomainParticipantResourceLimits::ignored_entity↩
_replacement_kind ( ) const
```

Getter (see setter with the same name)

8.92.2.115 remote_topic_query_allocation() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::remote_↩
topic_query_allocation (
            const  AllocationSettings & settings )
```

<<***extension***>> *(p. 116)* Allocation settings applied to remote TopicQueries.

Settings applied to the allocation of information about **rti::sub::TopicQuery** (p. 2516) objects created by other participants and discovered by this participant.

When the participant receives a new topic query that would make the current count go above `max_count`, it is not processed until the current count drops (i.e. another topic query is cancelled). The topic query stays in the Built-in ServiceRequest DataReader queue until it can be processed or it is cancelled.

[default] initial_count = 1; max_count = **dds::core::LENGTH_UNLIMITED** (p. 438); incremental_count = -1

[range] See allowed ranges in struct **rti::core::AllocationSettings** (p. 803)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Topic Queries** (p. 239)

8.92.2.116 remote_topic_query_allocation() [2/2]

```
AllocationSettings rti::core::policy::DomainParticipantResourceLimits::remote_topic_query_↩
allocation ( ) const
```

Getter (see setter with the same name)

8.92.2.117 remote_topic_query_hash_buckets() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::remote_↩
topic_query_hash_buckets (
            int32_t hash_buckets )
```

[DEPRECATED] (p. 296) <<***extension***>> *(p. 116)* Number of hash buckets for remote TopicQueries.

[default] 1

[range] [1, 10000]

See also

rti::core::policy::DomainParticipantResourceLimits::remote_topic_query_allocation (p. 1429)

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Topic Queries** (p. 239)

8.92.2.118 remote_topic_query_hash_buckets() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::remote_topic_query_hash_buckets ( )
const
```

Getter (see setter with the same name)

8.92.2.119 writer_data_tag_list_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::writer_↩
data_tag_list_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum number of data tags associated with a **dds::pub::DataWriter** (p. 1138).

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.120 writer_data_tag_list_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::writer_data_tag_list_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.121 writer_data_tag_string_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::writer_↩
data_tag_string_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum string length of the data tags associated with a **dds::pub::DataWriter** (p. 1138).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with the **dds::pub::DataWriter** (p. 1138) data tags.

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.122 writer_data_tag_string_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::writer_data_tag_string_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.123 reader_data_tag_list_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::reader_↩
data_tag_list_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum number of data tags associated with a **dds::sub::DataReader** (p. 976).

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.124 reader_data_tag_list_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::reader_data_tag_list_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.125 reader_data_tag_string_max_length() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::reader_↩
data_tag_string_max_length (
            int32_t max_length )
```

<<***extension***>> *(*p. *116)* Maximum string length of the data tags associated with a **dds::sub::DataReader** (p. 976).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with the **dds::sub::DataReader** (p. 976) data tags.

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.92.2.126 reader_data_tag_string_max_length() [2/2]

```
int32_t rti::core::policy::DomainParticipantResourceLimits::reader_data_tag_string_max_length ( )
const
```

Getter (see setter with the same name)

8.92.2.127 shmem_ref_transfer_mode_max_segments() [1/2]

```
DomainParticipantResourceLimits & rti::core::policy::DomainParticipantResourceLimits::shmem_ref↩
_transfer_mode_max_segments (
            uint32_t the_shmem_ref_transfer_mode_max_segments )
```

<<***extension***>> *(*p. *116)* Maximum number of segments created by all DataWriters belonging to a **dds::domain::↩ DomainParticipant** (p. 1326).

[default] 500

[range] [0,0xffffffff], but in practice, this value will be limited by the system-wide maximum number of shared memory segments. On a Linux machine, this value is provided by the kernel parameter shmmni.

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204), **Sending Large Data** (p. 197)

8.92.2.128 shmem_ref_transfer_mode_max_segments() [2/2]

```
uint32_t rti::core::policy::DomainParticipantResourceLimits::shmem_ref_transfer_mode_max_segments
( ) const
```

Getter (see setter with the same name)

8.93 dds::core::policy::Durability Class Reference

Specifies whether a **dds::pub::DataWriter** (p. 1138) will store and deliver previously published data samples to late-joining **dds::sub::DataReader** (p. 976) entities.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **Durability** ()

  *Creates the default **Durability** (p. 1433).*
- **Durability** (**dds::core::policy::DurabilityKind** the_kind)

  Creates an instance with a specific durability kind.
- **Durability** & **kind** (**dds::core::policy::DurabilityKind** the_kind)

  *Sets the **Durability** (p. 1433) kind.*
- **dds::core::policy::DurabilityKind kind** () const

  Getter (see setter with the same name)
- **dds::core::policy::Durability** & **direct_communication** (bool the_direct_communication)

  *<<**extension**>> (p. 116) Indicates whether or not a transient or persistent **dds::sub::DataReader** (p. 976) should receive samples directly from a transient or persistent **dds::pub::DataWriter** (p. 1138)*
- bool **direct_communication** () const

  *<<**extension**>> (p. 116) Getter (see setter with the same name)*
- **dds::core::policy::Durability** & **writer_depth** (int32_t the_writer_depth)

  *<<**extension**>> (p. 116) Indicates the number of samples a durable DataWriter will send to a late joining DataReader.*
- int32_t **writer_depth** () const

  *<<**extension**>> (p. 116) Getter (see setter with the same name)*
- **dds::core::policy::Durability** & **storage_settings** (const **rti::core::PersistentStorageSettings** &the_↩ storage_settings)

  *<<**extension**>> (p. 116) Used to configure durable writer history and durable reader state.*
- const **rti::core::PersistentStorageSettings** & **storage_settings** () const

  Gets the persistent storage settings by const-reference (see setter)
- **rti::core::PersistentStorageSettings** & **storage_settings** ()

  Gets the persistent storage settings by reference (see setter)

Static Public Member Functions

- static **Durability Volatile** ()

  *Creates a **Durability** (p. 1433) instance with volatile kind.*
- static **Durability TransientLocal** ()

  *Creates a **Durability** (p. 1433) instance with trasient-local kind.*
- static **Durability Transient** ()

  *Creates a **Durability** (p. 1433) instance with trasient kind.*
- static **Durability Persistent** ()

  *Creates a **Durability** (p. 1433) instance with persistent kind.*

8.93.1 Detailed Description

Specifies whether a **dds::pub::DataWriter** (p. 1138) will store and deliver previously published data samples to late-joining **dds::sub::DataReader** (p. 976) entities.

Entity:

dds::topic::Topic (p. 2474), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Status:

dds::core::status::StatusMask::offered_incompatible_qos() (p. 2380), **dds::core::status::StatusMask↩ ::requested_incompatible_qos()** (p. 2381)

Properties:

RxO (p. 514) = YES
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

See also

DURABILITY_SERVICE (p. 531)

8.93.2 Usage

It is possible for a **dds::pub::DataWriter** (p. 1138) to start publishing data before all (or any) **dds::sub::DataReader** (p. 976) entities have joined the network.

Moreover, a **dds::sub::DataReader** (p. 976) that joins the network after some data has been written could potentially be interested in accessing the most current values of the data, as well as potentially some history.

This policy makes it possible for a late-joining **dds::sub::DataReader** (p. 976) to obtain previously published samples.

By helping to ensure that DataReaders get all data that was sent by DataWriters, regardless of when it was sent, using this QoS policy can increase system tolerance to failure conditions.

Note that although related, this does not strictly control what data RTI Connext will maintain internally. That is, RTI Connext may choose to maintain some data for its own purposes (e.g., flow control) and yet not make it available to late-joining readers if the **DURABILITY** (p. 529) policy is set to **dds::core::policy::DurabilityKind_def::VOLATILE** (p. 1442).

8.93.2.1 Transient and Persistent Durability

For the purpose of implementing the DURABILITY QoS kind TRANSIENT or PERSISTENT, RTI Connext behaves *as if* for each Topic that has **dds::core::policy::Durability::kind** (p. 1437) of **dds::core::policy::DurabilityKind_def↩ ::TRANSIENT** (p. 1442) or **dds::core::policy::DurabilityKind_def::PERSISTENT** (p. 1442) there is a corresponding "built-in" **dds::sub::DataReader** (p. 976) and **dds::pub::DataWriter** (p. 1138) configured with the same DURABILITY kind. In other words, it is *as if* somewhere in the system, independent of the original **dds::pub::DataWriter** (p. 1138), there is a built-in durable **dds::sub::DataReader** (p. 976) subscribing to that Topic and a built-in durable DataWriter re-publishing it as needed for the new subscribers that join the system. This functionality is provided by the *RTI Persistence Service*.

The Persistence Service can configure itself based on the QoS of your application's **dds::pub::DataWriter** (p. 1138) and **dds::sub::DataReader** (p. 976) entities. For each transient or persistent **dds::topic::Topic** (p. 2474), the built-in fictitious Persistence Service **dds::sub::DataReader** (p. 976) and **dds::pub::DataWriter** (p. 1138) have their QoS configured from the QoS of your application's **dds::pub::DataWriter** (p. 1138) and **dds::sub::DataReader** (p. 976) entities that communicate on that **dds::topic::Topic** (p. 2474).

For a given **dds::topic::Topic** (p. 2474), the usual request/offered semantics apply to the matching between any **dds::pub::DataWriter** (p. 1138) in the domain that writes the **dds::topic::Topic** (p. 2474) and the built-in transient/persistent **dds::sub::DataReader** (p. 976) for that **dds::topic::Topic** (p. 2474); similarly for the built-in transient/persistent **dds::pub::DataWriter** (p. 1138) for a **dds::topic::Topic** (p. 2474) and any **dds::sub::DataReader** (p. 976) for the **dds::topic::Topic** (p. 2474). As a consequence, a **dds::pub::DataWriter** (p. 1138) that has an incompatible QoS will not send its data to the *RTI Persistence Service*, and a **dds::sub::DataReader** (p. 976) that has an incompatible QoS will not get data from it.

Incompatibilities between local **dds::sub::DataReader** (p. 976) and **dds::pub::DataWriter** (p. 1138) entities and the corresponding fictitious built-in transient/persistent entities cause the **dds::core::status::StatusMask::requested_incompatible_qos()** (p. 2381) and **dds::core::status::StatusMask::offered_incompatible_qos()** (p. 2380) to change and the corresponding **Listener** (p. 1637) invocations and/or signaling of **dds::core::cond::Condition** (p. 948) objects as they would with your application's own entities.

The value of **dds::core::policy::DurabilityService::service_cleanup_delay** (p. 1445) controls when *RTI Persistence Service* is able to remove all information regarding a data instances.

Information on a data instance is maintained until the following conditions are met:

1. The instance has been explicitly disposed (instance_state = NOT_ALIVE_DISPOSED),

and

1. All samples for the disposed instance have been acknowledged, including the dispose sample itself,

and

1. A time interval longer that **dds::core::policy::DurabilityService::service_cleanup_delay** (p. 1445) has elapsed since the moment RTI Connext detected that the previous two conditions were met. (Note: Only values of zero or **dds::core::Duration::infinite()** (p. 1451) are currently supported for the service_cleanup_delay)

The utility of **dds::core::policy::DurabilityService::service_cleanup_delay** (p. 1445) is apparent in the situation where an application disposes an instance and it crashes before it has a chance to complete additional tasks related to the disposition. Upon restart, the application may ask for initial data to regain its state and the delay introduced by the service_cleanup_delay will allow the restarted application to receive the information on the disposed instance and complete the interrupted tasks.

8.93.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered kind >= requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of DURABILITY kind are considered ordered such that **dds::core::policy::DurabilityKind_def::VOLATILE** (p. 1442) < **dds::core::policy::DurabilityKind_def::TRANSIENT_LOCAL** (p. 1442) < **dds::core::policy::DurabilityKind_def::TRANSIENT** (p. 1442) < **dds::core::policy::DurabilityKind_def::PERSISTENT** (p. 1442).

8.93.4 Constructor & Destructor Documentation

8.93.4.1 Durability() [1/2]

```
dds::core::policy::Durability::Durability ( )  [inline]
```

Creates the default **Durability** (p. 1433).

8.93.4.2 Durability() [2/2]

```
dds::core::policy::Durability::Durability (
            dds::core::policy::DurabilityKind the_kind )  [inline], [explicit]
```

Creates an instance with a specific durability kind.

*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of durability.

[default] dds::core::policy::DurabilityKind_def::VOLATILE (p. 1442)

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Durability** (p. 184), **Data Availability** (p. 167)

8.93.5 Member Function Documentation

8.93.5.1 kind() [1/2]

```
Durability & dds::core::policy::Durability::kind (
            dds::core::policy::DurabilityKind the_kind )  [inline]
```

Sets the **Durability** (p. 1433) kind.

*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The kind of durability.

[default] dds::core::policy::DurabilityKind_def::VOLATILE (p. 1442)

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Durability** (p. 184), **Data Availability** (p. 167)

8.93.5.2 kind() [2/2]

```
dds::core::policy::DurabilityKind dds::core::policy::Durability::kind ( ) const  [inline]
```

Getter (see setter with the same name)

8.93.5.3 Volatile()

```
static  Durability dds::core::policy::Durability::Volatile ( )  [inline], [static]
```

Creates a **Durability** (p. 1433) instance with volatile kind.

8.93.5.4 TransientLocal()

```
static  Durability dds::core::policy::Durability::TransientLocal ( )  [inline], [static]
```

Creates a **Durability** (p. 1433) instance with trasient-local kind.

8.93.5.5 Transient()

```
static  Durability dds::core::policy::Durability::Transient ( )  [inline], [static]
```

Creates a **Durability** (p. 1433) instance with trasient kind.

8.93.5.6 Persistent()

```
static  Durability dds::core::policy::Durability::Persistent ( )  [inline], [static]
```

Creates a **Durability** (p. 1433) instance with persistent kind.

8.93.5.7 direct_communication() [1/2]

```
dds::core::policy::Durability & direct_communication (
            bool the_direct_communication )
```

<<***extension***>> *(*p. *116)* Indicates whether or not a transient or persistent **dds::sub::DataReader** (p. 976) should receive samples directly from a transient or persistent **dds::pub::DataWriter** (p. 1138)

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

When direct_communication is set to true, a TRANSIENT or PERSISTENT **dds::sub::DataReader** (p. 976) will receive samples from both the original **dds::pub::DataWriter** (p. 1138) configured with TRANSIENT or PERSISTENT durability and the **dds::pub::DataWriter** (p. 1138) created by the persistence service. This peer-to-peer communication pattern provides low latency between end-points.

If the same sample is received from the original **dds::pub::DataWriter** (p. 1138) and the persistence service, the middleware will discard the duplicate.

When direct_communication is set to false, a TRANSIENT or PERSISTENT **dds::sub::DataReader** (p. 976) will only receive samples from the **dds::pub::DataWriter** (p. 1138) created by the persistence service. This brokered communication pattern provides a way to guarantee eventual consistency.

[default] true

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Durability** (p. 184), **Data Availability** (p. 167)

8.93.5.8 direct_communication() [2/2]

```
bool direct_communication ( ) const
```

<<***extension***>> *(*p. *116)* Getter (see setter with the same name)

8.93.5.9 writer_depth() [1/2]

```
dds::core::policy::Durability & writer_depth (
            int32_t the_writer_depth )
```

<<***extension***>> *(p. 116)* Indicates the number of samples a durable DataWriter will send to a late joining DataReader.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The default value, **rti::core::policy::auto_writer_depth()** (p. 531), makes this parameter equal to the following:

- **dds::core::policy::History::depth** (p. 1605) if the history kind is **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607).
- **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207) in the **dds::core::policy::ResourceLimits** (p. 2203) if the history kind is **dds::core::policy::HistoryKind_def::KEEP_ALL** (p. 1607).

The `writer_depth` must be <= **dds::core::policy::History::depth** (p. 1605).

`writer_depth` applies only to non-volatile DataWriters (those for which the kind is TRANSIENT_LOCAL, TRANSIENT, or PERSISTENT).

When **rti::core::policy::Batch::enable** (p. 882) is set to true, `writer_depth` acts as a minimum number of samples per instance that will be sent to late joiners, as opposed to the maximum. As long as a batch contains a single sample that falls within the `writer_depth` for the instance to which it belongs, then the entire batch will be sent. This means that batches sent to late-joining DataReaders may contain more samples per instance than is specified by the `writer_depth` QoS setting.

When a DataWriter responds to a TopicQuery, the samples that are evaluated against the TopicQuery filter are only those samples that fall within the `writer_depth`, not the **dds::core::policy::History::depth** (p. 1605).

Setting `writer_depth` on the DataReader side will be ignored.

[default] rti::core::policy::auto_writer_depth() (p. 531)

Categories

Immutable (p. 243), **Durability** (p. 184), **Applied Per-Instance** (p. 287), **Data Availability** (p. 167)

8.93.5.10 writer_depth() [2/2]

```
int32_t writer_depth ( ) const
```

<<***extension***>> *(p. 116)* Getter (see setter with the same name)

8.93.5.11 storage_settings() [1/3]

```
dds::core::policy::Durability & storage_settings (
            const  rti::core::PersistentStorageSettings & the_storage_settings )
```

<<***extension***>> *(p. 116)* Used to configure durable writer history and durable reader state.

By default, durable writer history and durable reader state are disabled. This means that a DataWriter will not persist its historical cache and a DataReader will not persist its state.

To enable durable writer history and durable reader state set **rti::core::PersistentStorageSettings::enable** (p. 1929) to true.

Categories

Immutable (p. 243), **Durability** (p. 184), **Data Availability** (p. 167)

8.93.5.12 storage_settings() [2/3]

```
const  rti::core::PersistentStorageSettings & storage_settings ( ) const
```

Gets the persistent storage settings by const-reference (see setter)

8.93.5.13 storage_settings() [3/3]

```
rti::core::PersistentStorageSettings & storage_settings ( )
```

Gets the persistent storage settings by reference (see setter)

8.94 dds::core::policy::DurabilityKind_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) DurabilityKind.

```
#include <PolicyKind.hpp>
```

Public Types

- enum **type** {
  VOLATILE ,
  TRANSIENT_LOCAL ,
  TRANSIENT ,
  PERSISTENT }

  The underlying `enum` *type.*

8.94.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) DurabilityKind.

8.94.2 Member Enumeration Documentation

8.94.2.1 type

`enum` **dds::core::policy::DurabilityKind_def::type**

The underlying `enum` type.

Enumerator

VOLATILE	**[default]** RTI Connext does not need to keep any samples of data instances on behalf of any **dds::sub::DataReader** (p. 976) that is unknown by the **dds::pub::DataWriter** (p. 1138) at the time the instance is written. In other words, RTI Connext will only attempt to provide the data to existing subscribers. This option does not require RTI Persistence Service.
TRANSIENT_LOCAL	RTI Connext will attempt to keep some samples so that they can be delivered to any potential late-joining **dds::sub::DataReader** (p. 976). Which particular samples are kept depends on other QoS such as **dds::core::policy::History** (p. 1602) and **dds::core::policy::ResourceLimits** (p. 2203). RTI Connext is only required to keep the data in memory of the **dds::pub::DataWriter** (p. 1138) that wrote the data. Data is not required to survive the **dds::pub::DataWriter** (p. 1138). For this setting to be effective, you must also set the **dds::core::policy::Reliability::kind** (p. 2153) to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157). This option does not require RTI Persistence Service.
TRANSIENT	RTI Connext will attempt to keep some samples so that they can be delivered to any potential late-joining **dds::sub::DataReader** (p. 976). Which particular samples are kept depends on other QoS such as **dds::core::policy::History** (p. 1602) and **dds::core::policy::ResourceLimits** (p. 2203). RTI Connext is only required to keep the data in memory and not in permanent storage. Data is not tied to the lifecycle of the **dds::pub::DataWriter** (p. 1138). Data will survive the **dds::pub::DataWriter** (p. 1138). This option requires RTI Persistence Service.
PERSISTENT	Data is kept on permanent storage, so that they can outlive a system session. This option requires RTI Persistence Service.

8.95 dds::core::policy::DurabilityService Class Reference

Configures the external RTI Persistence Service used by persistent and transient DataWriters.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **DurabilityService** ()

  Creates the default policy.
- **DurabilityService** (const **dds::core::Duration** &the_service_cleanup_delay, **dds::core::policy::HistoryKind** the_history_kind, int32_t the_history_depth, int32_t the_max_samples, int32_t the_max_instances, int32_t the_max_samples_per_instance)

  Creates an instance with all the parameters set.
- **DurabilityService** & **service_cleanup_delay** (const **dds::core::Duration** &d)

  *<<**spec**>> (p. 112) Controls when the service is able to remove all information regarding a data instances.*
- **dds::core::Duration service_cleanup_delay** () const

  Getter (see setter with the same name)
- **DurabilityService** & **history_kind** (**dds::core::policy::HistoryKind** the_kind)

  *<<**spec**>> (p. 112) The kind of history to apply in recouping durable data.*
- **dds::core::policy::HistoryKind history_kind** () const

  Getter (see setter with the same name)
- **DurabilityService** & **history_depth** (int32_t the_depth)

  *<<**spec**>> (p. 112) Setting to use for the **dds::core::policy::Durability::writer_depth** (p. 1439) when recouping durable data.*
- int32_t **history_depth** () const

  Getter (see setter with the same name)
- **DurabilityService** & **max_samples** (int32_t the_max_samples)

  *<<**spec**>> (p. 112) Part of resource limits QoS policy to apply when feeding a late joiner.*
- int32_t **max_samples** () const

  Getter (see setter with the same name)
- **DurabilityService** & **max_instances** (int32_t the_max_instances)

  *<<**spec**>> (p. 112) Part of resource limits QoS policy to apply when feeding a late joiner.*
- **DurabilityService** & **max_samples_per_instance** (int32_t the_max_samples_per_instance)

  *<<**spec**>> (p. 112) Part of resource limits QoS policy to apply when feeding a late joiner.*
- int32_t **max_samples_per_instance** () const

  Getter (see setter with the same name)

8.95.1 Detailed Description

Configures the external RTI Persistence Service used by persistent and transient DataWriters.

Entity:

dds::topic::Topic (p. 2474), **dds::pub::DataWriter** (p. 1138)

Properties:

RxO (p. 514) = NO
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

See also

DURABILITY (p. 529)

HISTORY (p. 533)

RESOURCE_LIMITS (p. 546)

8.95.2 Usage

When a DataWriter's **dds::core::policy::Durability::kind** (p. 1437) is **dds::core::policy::DurabilityKind_def::↩ PERSISTENT** (p. 1442) or **dds::core::policy::DurabilityKind_def::TRANSIENT** (p. 1442), an external service, the *RTI Persistence Service*, is used to store and possibly forward the data sent by the **dds::pub::DataWriter** (p. 1138) to **dds::sub::DataReader** (p. 976) objects that are created *after* the data was initially sent.

This QoS policy is used to configure certain parameters of the Persistence Service when it operates on the behalf of the **dds::pub::DataWriter** (p. 1138), such as how much data to store. For example, it configures the **DURABILITY** (p. 529), **HISTORY** (p. 533), and the **RESOURCE_LIMITS** (p. 546) used by the fictitious DataReader and DataWriter used by the Persistence Service. Note, however, that the Persistence Service itself may be configured to ignore these values and instead use values from its own configuration file.

8.95.3 Constructor & Destructor Documentation

8.95.3.1 DurabilityService() [1/2]

```
dds::core::policy::DurabilityService::DurabilityService ( )  [inline]
```

Creates the default policy.

8.95.3.2 DurabilityService() [2/2]

```
dds::core::policy::DurabilityService::DurabilityService (
            const  dds::core::Duration & the_service_cleanup_delay,
             dds::core::policy::HistoryKind the_history_kind,
            int32_t the_history_depth,
            int32_t the_max_samples,
            int32_t the_max_instances,
            int32_t the_max_samples_per_instance )  [inline]
```

Creates an instance with all the parameters set.

8.95.4 Member Function Documentation

8.95.4.1 service_cleanup_delay() [1/2]

```
DurabilityService & dds::core::policy::DurabilityService::service_cleanup_delay (
            const  dds::core::Duration & d )  [inline]
```

<<***spec***>> *(p. 112)* Controls when the service is able to remove all information regarding a data instances.

When the service cleanup delay is set to 0, disposed instances will be completely removed from the service. Only values of 0 and **dds::core::Duration::infinite()** (p. 1451) are currently supported.

[default] 0

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Durability** (p. 184), **Instance Lifecycle** (p. 193), **Data Availability** (p. 167)

8.95.4.2 service_cleanup_delay() [2/2]

```
dds::core::Duration dds::core::policy::DurabilityService::service_cleanup_delay ( ) const  [inline]
```

Getter (see setter with the same name)

8.95.4.3 history_kind() [1/2]

```
DurabilityService & dds::core::policy::DurabilityService::history_kind (
             dds::core::policy::HistoryKind the_kind )  [inline]
```

<<***spec***>> *(p. 112)* The kind of history to apply in recouping durable data.

[default] dds::core::policy::HistoryKind_def::KEEP_LAST (p. 1607)

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Durability** (p. 184), **Applied Per-Instance** (p. 287), **Sample Memory Management** (p. 230)

8.95.4.4 history_kind() [2/2]

```
dds::core::policy::HistoryKind dds::core::policy::DurabilityService::history_kind ( ) const  [inline]
```

Getter (see setter with the same name)

8.95.4.5 history_depth() [1/2]

```
DurabilityService & dds::core::policy::DurabilityService::history_depth (
            int32_t the_depth )  [inline]
```

<<***spec***>> *(*p. *112)* Setting to use for the **dds::core::policy::Durability::writer_depth** (p. 1439) when recouping durable data.

If the **dds::core::policy::History::depth** (p. 1605) is set to a value lower than this value, **dds::core::policy::History::depth** (p. 1605) will be set equal to the value of this field.

[default] rti::core::policy::auto_writer_depth() (p. 531) (1)

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Durability** (p. 184), **Applied Per-Instance** (p. 287), **Sample Memory Management** (p. 230)

8.95.4.6 history_depth() [2/2]

```
int32_t dds::core::policy::DurabilityService::history_depth ( ) const  [inline]
```

Getter (see setter with the same name)

8.95.4.7 max_samples() [1/2]

```
DurabilityService & dds::core::policy::DurabilityService::max_samples (
            int32_t the_max_samples )  [inline]
```

<<***spec***>> *(*p. *112)* Part of resource limits QoS policy to apply when feeding a late joiner.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Durability** (p. 184), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Data Availability** (p. 167)

8.95.4.8 max_samples() [2/2]

```
int32_t dds::core::policy::DurabilityService::max_samples ( ) const  [inline]
```

Getter (see setter with the same name)

8.95.4.9 max_instances()

```
DurabilityService & dds::core::policy::DurabilityService::max_instances (
            int32_t the_max_instances )  [inline]
```

<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Durability** (p. 184), **Instance Lifecycle** (p. 193), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Data Availability** (p. 167)

8.95.4.10 max_samples_per_instance() [1/2]

```
DurabilityService & dds::core::policy::DurabilityService::max_samples_per_instance (
            int32_t the_max_samples_per_instance )  [inline]
```

<<***spec***>> *(p. 112)* Part of resource limits QoS policy to apply when feeding a late joiner.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Durability** (p. 184), **Instance Lifecycle** (p. 193), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Data Availability** (p. 167)

8.95.4.11 max_samples_per_instance() [2/2]

```
int32_t dds::core::policy::DurabilityService::max_samples_per_instance ( ) const  [inline]
```

Getter (see setter with the same name)

8.96 dds::core::Duration Class Reference

<<***value-type***>> *(*p. *350)* Represents a time interval

`#include <Duration.hpp>`

Public Member Functions

- **Duration** ()

  Create a ***Duration*** *(*p. *1448) elapsing zero seconds.*
- **Duration** (int32_t **sec**, uint32_t **nanosec**=0)

  Create a duration elapsing a specific amount of time.
- template<typename Rep , typename Period >
  Duration (const std::chrono::duration< Rep, Period > &duration)

  <<***C++11***>> *(*p. *353)* <<***extension***>> *(*p. *354) Allow implicit creation from std::chrono::duration*
- int32_t **sec** () const

  Get the number of seconds represented by this ***Duration*** *(*p. *1448) object.*
- void **sec** (int32_t s)

  Set the number of seconds represented by this ***Duration*** *(*p. *1448) object.*
- uint32_t **nanosec** () const

  Get the number of nanoseconds represented by this ***Duration*** *(*p. *1448) object.*
- void **nanosec** (uint32_t ns)

  Set the number of nanoseconds represented by this ***Duration*** *(*p. *1448) object.*
- int **compare** (const **Duration** &that) const

  Compare two ***Duration*** *(*p. *1448) objects.*
- bool **operator>** (const **Duration** &that) const

  Check if this ***Duration*** *(*p. *1448) is greater than another.*
- bool **operator>=** (const **Duration** &that) const

  Check if this ***Duration*** *(*p. *1448) is greater than or equal another.*
- bool **operator==** (const **Duration** &that) const

  Check if this ***Duration*** *(*p. *1448) is equal to another.*
- bool **operator!=** (const **Duration** &other) const

  Check if this ***Duration*** *(*p. *1448) is not equal to another.*
- bool **operator<=** (const **Duration** &that) const

  Check if this ***Duration*** *(*p. *1448) is less than or equal another.*
- bool **operator<** (const **Duration** &that) const

  Check if this ***Duration*** *(*p. *1448) is less than another.*
- **Duration** & **operator+=** (const **Duration** &a_ti)

  Add a ***Duration*** *(*p. *1448) to this* ***Duration*** *(*p. *1448).*
- **Duration** & **operator-=** (const **Duration** &a_ti)

  Subtract a ***Duration*** *(*p. *1448) from this* ***Duration*** *(*p. *1448).*
- **Duration operator+** (const **Duration** &other) const

  Add two ***Duration*** *(*p. *1448) objects.*
- **Duration operator-** (const **Duration** &other) const

  Subtract a ***Duration*** *(*p. *1448).*
- uint64_t **to_nanosecs** () const

*Returns this **Duration** (p. 1448) in nanoseconds.*

- uint64_t **to_millisecs** () const

  *Returns this **Duration** (p. 1448) in milliseconds.*

- uint64_t **to_microsecs** () const

  *Returns this **Duration** (p. 1448) in microseconds.*

- double **to_secs** () const

  *Returns this **Duration** (p. 1448) in seconds.*

- std::chrono::nanoseconds **to_chrono** () const

  *<<**C++11**>> (p. 353) <<**extension**>> (p. 354) Converts to std::chrono::nanoseconds*

Static Public Member Functions

- static **Duration** **zero** ()

  Returns a zero duration.

- static **Duration** **infinite** ()

  *Special value that represents an infinite **Duration** (p. 1448).*

- static **Duration** **automatic** ()

  Special value that indicates that RTI Connext will automatically assign a value.

- static **Duration** **from_microsecs** (uint64_t microseconds)

  *Create a **Duration** (p. 1448) elapsing a specific number of microseconds.*

- static **Duration** **from_millisecs** (uint64_t milliseconds)

  *Create a **Duration** (p. 1448) elapsing a specific number of milliseconds.*

- static **Duration** **from_secs** (double seconds)

  *Create a **Duration** (p. 1448) elapsing a specific number of seconds.*

Related Functions

(Note that these are not member functions.)

- **Duration** **operator*** (uint32_t lhs, const **Duration** &rhs)

  *Multiply a **Duration** (p. 1448) object by an unsigned integer.*

- **Duration** **operator*** (const **Duration** &lhs, uint32_t rhs)

  *Multiply a **Duration** (p. 1448) object by an unsigned integer.*

- **Duration** **operator/** (const **Duration** &lhs, uint32_t rhs)

  *Divide a **Duration** (p. 1448) object by an unsigned integer.*

- void **swap** (**Duration** &lhs, **Duration** &rhs) OMG_NOEXCEPT

  *Swap the contents of two **Duration** (p. 1448) objects.*

8.96.1 Detailed Description

*<<**value-type**>> (p. 350)* Represents a time interval

Examples

Foo_publisher.cxx, and **Foo_subscriber.cxx**.

8.96.2 Constructor & Destructor Documentation

8.96.2.1 Duration() [1/3]

```
dds::core::Duration::Duration ( )
```

Create a **Duration** (p. 1448) elapsing zero seconds.

8.96.2.2 Duration() [2/3]

```
dds::core::Duration::Duration (
            int32_t sec,
            uint32_t nanosec = 0 )  [explicit]
```

Create a duration elapsing a specific amount of time.

Parameters

sec	The number of seconds to represent
nanosec	The number of nanoseconds to represent

8.96.2.3 Duration() [3/3]

```
template<typename Rep , typename Period >
dds::core::Duration::Duration (
            const std::chrono::duration< Rep, Period > & duration )  [inline]
```

<<***C++11***>> *(p. 353)* <<***extension***>> *(p. 354)* Allow implicit creation from std::chrono::duration

For example:

```
dds::core::cond::WaitSet waitset;
// ...
waitset.wait(std::chrono::seconds(1) + std::chrono::milliseconds(250));
```

8.96.3 Member Function Documentation

8.96.3.1 zero()

```
static Duration dds::core::Duration::zero ( ) [static]
```

Returns a zero duration.

Returns

Duration() (p. 1450)

8.96.3.2 infinite()

```
static Duration dds::core::Duration::infinite ( ) [static]
```

Special value that represents an infinite **Duration** (p. 1448).

8.96.3.3 automatic()

```
static Duration dds::core::Duration::automatic ( ) [static]
```

Special value that indicates that RTI Connext will automatically assign a value.

8.96.3.4 from_microsecs()

```
static Duration dds::core::Duration::from_microsecs (
            uint64_t microseconds ) [static]
```

Create a **Duration** (p. 1448) elapsing a specific number of microseconds.

Parameters

microseconds	The number of microseconds to construct the object from

Returns

A newly constructed **Duration** (p. 1448) object

8.96.3.5 from_millisecs()

```
static Duration dds::core::Duration::from_millisecs (
            uint64_t milliseconds ) [static]
```

Create a **Duration** (p. 1448) elapsing a specific number of milliseconds.

Parameters

milliseconds	The number of milliseconds to construct the object from

Returns

A newly constructed **Duration** (p. 1448) object

Referenced by **dds::core::policy::DestinationOrder::ReceptionTimestamp()**.

8.96.3.6 from_secs()

```
static Duration dds::core::Duration::from_secs (
            double seconds ) [static]
```

Create a **Duration** (p. 1448) elapsing a specific number of seconds.

Parameters

seconds	The number of seconds to construct the object from

Returns

A newly constructed **Duration** (p. 1448) object

8.96.3.7 sec() [1/2]

```
int32_t dds::core::Duration::sec ( ) const
```

Get the number of seconds represented by this **Duration** (p. 1448) object.

Returns

The number of seconds (excluding the nanoseconds)

8.96.3.8 sec() [2/2]

```
void dds::core::Duration::sec (
            int32_t s )
```

Set the number of seconds represented by this **Duration** (p. 1448) object.

Parameters

s	The number of seconds to set

8.96.3.9 nanosec() [1/2]

```
uint32_t dds::core::Duration::nanosec ( ) const
```

Get the number of nanoseconds represented by this **Duration** (p. 1448) object.

Returns

The number of nanoseconds (excluding the seconds)

8.96.3.10 nanosec() [2/2]

```
void dds::core::Duration::nanosec (
            uint32_t ns )
```

Set the number of nanoseconds represented by this **Duration** (p. 1448) object.

Parameters

ns	The number of nanoseconds to set

8.96.3.11 compare()

```
int dds::core::Duration::compare (
            const  Duration & that ) const
```

Compare two **Duration** (p. 1448) objects.

Parameters

that	The **Duration** (p. 1448) object to compare with this **Duration** (p. 1448).

Returns

int The result of the comparison can be: -1 if this **Duration** (p. 1448) is less than other; 0 if they are equal; 1 if this **Duration** (p. 1448) is greater than other

8.96.3.12 operator>()

```
bool dds::core::Duration::operator> (
            const  Duration & that ) const
```

Check if this **Duration** (p. 1448) is greater than another.

Parameters

that	The **Duration** (p. 1448) to compare with this **Duration** (p. 1448).

Returns

bool false if this **Duration** (p. 1448) is less than or equal to the other object, true otherwise.

8.96.3.13 operator>=()

```
bool dds::core::Duration::operator>= (
            const  Duration & that ) const
```

Check if this **Duration** (p. 1448) is greater than or equal another.

Parameters

that	The **Duration** (p. 1448) to compare with this **Duration** (p. 1448).

Returns

bool false if this **Duration** (p. 1448) is less than to the other object, true otherwise.

8.96.3.14 operator==()

```
bool dds::core::Duration::operator== (
            const  Duration & that ) const
```

Check if this **Duration** (p. 1448) is equal to another.

Parameters

that	The **Duration** (p. 1448) to compare with this **Duration** (p. 1448).

Returns

bool false if this **Duration** (p. 1448) is not equal to the other object, true otherwise.

8.96.3.15 operator"!=()

```
bool dds::core::Duration::operator!= (
            const  Duration & other ) const
```

Check if this **Duration** (p. 1448) is not equal to another.

Parameters

other	The **Duration** (p. 1448) to compare with this **Duration** (p. 1448).

Returns

bool false if this **Duration** (p. 1448) is equal to the other object, true otherwise.

8.96.3.16 operator<=()

```
bool dds::core::Duration::operator<= (
            const  Duration & that ) const
```

Check if this **Duration** (p. 1448) is less than or equal another.

Parameters

that	The **Duration** (p. 1448) to compare with this **Duration** (p. 1448).

Returns

bool false if this **Duration** (p. 1448) is greater than to the other object, true otherwise.

8.96.3.17 operator<()

```
bool dds::core::Duration::operator< (
            const  Duration & that ) const
```

Check if this **Duration** (p. 1448) is less than another.

Parameters

that	The **Duration** (p. 1448) to compare with this **Duration** (p. 1448).

Returns

bool false if this **Duration** (p. 1448) is greater than or equal to the other object, true otherwise.

8.96.3.18 operator+=()

```
Duration & dds::core::Duration::operator+= (
            const  Duration & a_ti )
```

Add a **Duration** (p. 1448) to this **Duration** (p. 1448).

Parameters

a_ti	The **Duration** (p. 1448) to add

Returns

Duration (p. 1448)& This **Duration** (p. 1448), with the added **Duration** (p. 1448)

8.96.3.19 operator-=()

```
Duration & dds::core::Duration::operator-= (
            const  Duration & a_ti )
```

Subtract a **Duration** (p. 1448) from this **Duration** (p. 1448).

Parameters

a↩ _ti	The **Duration** (p. 1448) to subtract

Returns

Duration (p. 1448)& This **Duration** (p. 1448), after the subtraction

8.96.3.20 operator+()

```
Duration dds::core::Duration::operator+ (
            const  Duration & other ) const
```

Add two **Duration** (p. 1448) objects.

Parameters

other	The other **Duration** (p. 1448) to add to this one

Returns

The result of the addition

8.96.3.21 operator-()

```
Duration dds::core::Duration::operator- (
            const  Duration & other ) const
```

Subtract a **Duration** (p. 1448).

Parameters

other	The **Duration** (p. 1448) to subract from this one

Returns

The result of the subtraction

8.96.3.22 to_nanosecs()

```
uint64_t dds::core::Duration::to_nanosecs ( ) const
```

Returns this **Duration** (p. 1448) in nanoseconds.

Returns

the **Duration** (p. 1448) in nanoseconds

8.96.3.23 to_millisecs()

```
uint64_t dds::core::Duration::to_millisecs ( ) const
```

Returns this **Duration** (p. 1448) in milliseconds.

Returns

the **Duration** (p. 1448) in milliseconds

8.96.3.24 to_microsecs()

```
uint64_t dds::core::Duration::to_microsecs ( ) const
```

Returns this **Duration** (p. 1448) in microseconds.

Returns

the **Duration** (p. 1448) in microseconds

8.96.3.25 to_secs()

```
double dds::core::Duration::to_secs ( ) const
```

Returns this **Duration** (p. 1448) in seconds.

Returns

the **Duration** (p. 1448) in seconds

8.96.3.26 to_chrono()

```
std::chrono::nanoseconds dds::core::Duration::to_chrono ( ) const  [inline]
```

<<***C++11***>> *(p. 353)* <<***extension***>> *(p. 354)* Converts to std::chrono::nanoseconds

Returns

The **Duration** (p. 1448) in nanoseconds.

8.96.4 Friends And Related Function Documentation

8.96.4.1 operator*() [1/2]

```
Duration operator* (
            uint32_t lhs,
            const  Duration & rhs )  [related]
```

Multiply a **Duration** (p. 1448) object by an unsigned integer.

Parameters

lhs	The unsigned integer to multiply the **Duration** (p. 1448) object by
rhs	The **Duration** (p. 1448) object to multiply

Returns

Duration (p. 1448) The result of the multiplication

8.96.4.2 operator*() [2/2]

```
Duration operator* (
            const  Duration & lhs,
            uint32_t rhs )  [related]
```

Multiply a **Duration** (p. 1448) object by an unsigned integer.

Parameters

lhs	The **Duration** (p. 1448) object to multiply
rhs	The unsigned integer to multiply the **Duration** (p. 1448) object by

Returns

Duration (p. 1448) The result of the multiplication

8.96.4.3 operator/()

```
Duration operator/ (
            const  Duration & lhs,
            uint32_t rhs )  [related]
```

Divide a **Duration** (p. 1448) object by an unsigned integer.

Parameters

lhs	The dividend
rhs	The divisor

Returns

Duration (p. 1448) The result of dividing the two **Duration** (p. 1448) objects

8.96.4.4 swap()

```
void swap (
             Duration & lhs,
             Duration & rhs )  [related]
```

Swap the contents of two **Duration** (p. 1448) objects.

Parameters

lhs	One of the **Duration** (p. 1448) objects
rhs	One of the **Duration** (p. 1448) objects

Examples

Foo.hpp.

Referenced by **dds::core::basic_string< CharType, Allocator >::basic_string()**, and **dds::core::basic_string< CharType, Allocator >::operator=()**.

8.97 rti::topic::dynamic_type< TopicType > Struct Template Reference

Provides a DynamicType that represents an IDL-generated type.

`#include <rti/topic/TopicTraits.hpp>`

8.97.1 Detailed Description

template<typename TopicType>
struct rti::topic::dynamic_type< TopicType >

Provides a DynamicType that represents an IDL-generated type.

This type is specialized for **IDL-generated types** (p. 606) to provide a **`get()`** (p. 820) function whose return type is a const reference to a concrete subclass of **dds::core::xtypes::DynamicType** (p. 1504), such as **dds::core::xtypes::StructType** (p. 2400), **dds::core::xtypes::UnionType** (p. 2584), **dds::core::xtypes::EnumType** (p. 1535).

For example, given the following IDL type **`Foo`** (p. 1588):

```
struct Foo {
    long x;
};
```

You can obtain its DynamicType (in this case a StructType) as follows:

```
const dds::core::xtypes::StructType& foo_dynamic = rti::topic::dynamic_type<Foo>::get();
std::cout « foo_dynamic.name() « std::endl; // Output: Foo
```

Note that if the type is an IDL alias such as the following:

```
typedef Foo Bar;
```

Then `rti::topic::dynamic_type<Bar>` resolves to `rti::topic::dynamic_type<Foo>`, both returning the same `StructType`. If you want to explicitly access the `AliasType` use the special tag type `Bar_AliasTag_t`:

```
const dds::core::xtypes::AliasType& bar_dynamic = rti::topic::dynamic_type<Bar_AliasTag_t>::get();
std::cout « bar_dynamic.name() « std::endl; // Output: Bar
```

See also

8.98 dds::core::xtypes::DynamicData Class Reference

<<***value-type***>> *(p. 350)* A data sample of any complex data type, which can be inspected and manipulated reflectively.

`#include <dds/core/xtypes/DynamicData.hpp>`

Public Member Functions

- **DynamicData** (const **dds::core::xtypes::DynamicType** & **type**)

  *Creates a **DynamicData*** (p. *1461*) *instance for a type.*
- **DynamicData** (const **dds::core::xtypes::DynamicType** & **type**, const DynamicDataProperty &property)

  *Creates a **DynamicData*** (p. *1461*) *instance for a type with specific memory-management.*
- template<typename T >
  void **value** (const std::string &name, const T &v)

  Sets the value of a member by member name.
- template<typename T >
  void **value** (uint32_t index, const T &v)

  Sets the value of a member by member index.
- template<typename T >
  T **value** (const std::string &name) const

  Gets the value of a member by member name.
- template<typename T >
  T **value** (uint32_t index) const

  Gets the value of a member by member index.
- template<typename T >
  void **get_values** (const std::string &name, std::vector< T > &out_array) const

  Obtains the values of an array or sequence member by member name.
- template<typename T >
  void **get_values** (uint32_t index, std::vector< T > &out_array) const

  Obtains the values of an array or sequence member by member index.
- template<typename T >
  std::vector< T > **get_values** (uint32_t index) const

  Obtains the values of an array or sequence member by member index.
- template<typename T >
  std::vector< T > **get_values** (const std::string &name) const

  Obtains the values of an array or sequence member by member name.
- template<typename T >
  void **set_values** (uint32_t index, const std::vector< T > &v)

  Sets the values of an array or sequence member by member index.
- template<typename T >
  void **set_values** (const std::string &name, const std::vector< T > &v)

  Sets the values of an array or sequence member by member name.
- LoanedDynamicData **loan_value** (const std::string &name)

  Gets a view of a complex member.
- LoanedDynamicData **loan_value** (uint32_t index)

  Gets a view of a complex member.
- LoanedDynamicData & **loan_value** (LoanedDynamicData &data, const std::string &name)

  Gets a view of a complex member.
- LoanedDynamicData & **loan_value** (LoanedDynamicData &data, uint32_t mid)

  Gets a view of a complex member.
- uint32_t **discriminator_value** () const

  *Deprecated. Use **get_discriminator()*** (p. *1478*) *instead. Obtains the value of the union discriminator (**UnionType*** (p. *2584*) *only)*
- int32_t **get_discriminator** () const

  *Obtains the value of the union discriminator (**UnionType*** (p. *2584*) *only)*

- void **set_discriminator** (int32_t disc)

  *Sets the value of the union discriminator (**UnionType** (p. 2584) only)*
- void **clear_all_members** ()

  Clear the contents of all data members of this object, including key members.
- void **clear_optional_member** (const std::string &name)

  Clear the contents of a single optional data member of this object.
- void **clear_optional_member** (uint32_t index)

  Clear the contents of a single optional data member of this object.
- void **clear_member** (const std::string &name)

  Clear the contents of a single data member of this object.
- void **clear_member** (uint32_t index)

  Clear the contents of a single data member of this object.
- bool **is_cdr** () const

  *Determines if this **DynamicData** (p. 1461) sample is represented as CDR.*
- std::pair< const char *, uint32_t > **get_cdr_buffer** () const

  *Gets the CDR buffer of this **DynamicData** (p. 1461) sample.*
- void **set_cdr_buffer** (const char *buffer, uint32_t length)

  *Sets the CDR buffer of this **DynamicData** (p. 1461) sample.*
- const **dds::core::xtypes::DynamicType** & **type** () const

  *Gets the data type of this **DynamicData** (p. 1461).*
- **dds::core::xtypes::TypeKind type_kind** () const

  *Gets the data type kind of this **DynamicData** (p. 1461).*
- uint32_t **member_count** () const

  Get the number of members in this sample.
- bool **member_exists** (const std::string &name) const

  Indicates whether a member exists in this sample.
- bool **member_exists** (uint32_t index) const

  Indicates whether a member exists in this sample.
- bool **member_exists_in_type** (const std::string &name) const

  *Indicates whether a member with a particular name exists in **type()** (p. 1482)*
- bool **member_exists_in_type** (uint32_t index) const

  *Indicates whether a member of a particular index exists in **type()** (p. 1482)*
- DynamicDataInfo **info** () const

  Returns information about this sample.
- **rti::core::xtypes::DynamicDataMemberInfo member_info** (const std::string &name) const

  Returns information about a member.
- **rti::core::xtypes::DynamicDataMemberInfo member_info** (uint32_t index) const

  Returns information about a member.
- uint32_t **member_index** (const std::string &name) const

  Translates from member name to member index.
- bool **is_member_key** (const std::string &name) const

  Returns whether a member is a key.
- bool **is_member_key** (uint32_t index) const

  Returns whether a member is a key.
- **dds::core::xtypes::DynamicType member_type** (const std::string &name) const

  Returns the type of a member.
- **dds::core::xtypes::DynamicType member_type** (uint32_t index) const

Returns the type of a member.

- void **from_string** (RTI_STRING_IN str, **rti::topic::PrintFormatKind** format_kind=rti::topic::PrintFormatKind::↩ JSON)

  *Populates this **DynamicData** (p. 1461) object from a JSON string representation.*

Related Functions

(Note that these are not member functions.)

- std::vector< char > & **to_cdr_buffer** (std::vector< char > &buffer, const **DynamicData** &sample, **dds::core↩ ::policy::DataRepresentationId** representation_id= **dds::core::policy::DataRepresentation::auto_id**())

  *<<**extension**>> (p. 354) Serializes a **DynamicData** (p. 1461) sample into CDR format*
- **DynamicData** & **from_cdr_buffer** (**DynamicData** &sample, const std::vector< char > &buffer)

  *<<**extension**>> (p. 354) Creates a **DynamicData** (p. 1461) sample by deserializing a CDR buffer'*
- template<typename TopicType >
  TopicType **convert** (const **DynamicData** &sample)

  *<<**extension**>> (p. 354) Creates a typed sample from a **DynamicData** (p. 1461) sample.*
- template<typename TopicType >
  DynamicData **convert** (const TopicType &sample)

  *<<**extension**>> (p. 354) Creates a **DynamicData** (p. 1461) sample from a typed sample*
- template<typename TopicType >
  bool **can_convert** (const **DynamicData** &sample)

  *<<**extension**>> (p. 354) Determines if the **DynamicType** (p. 1504) of this sample is equal to the **DynamicType** (p. 1504) of `TopicType`*
- template<typename... Types>
  std::tuple< Types... > **get_tuple** (const **dds::core::xtypes::DynamicData** &data)

  *<<**C++11**>> (p. 353) <<**experimental**>> (p. 355) <<**extension**>> (p. 354) Retrieves all the values of a **Dynamic↩ Data** (p. 1461) object into an std::tuple at once*
- template<typename... Types>
  void **set_tuple** (**dds::core::xtypes::DynamicData** &data, const std::tuple< Types... > & **value**)

  *<<**C++11**>> (p. 353) <<**experimental**>> (p. 355) <<**extension**>> (p. 354) Assigns the values of a std::tuple to a **DynamicData** (p. 1461) object at once*

8.98.1 Detailed Description

*<<**value-type**>> (p. 350)* A data sample of any complex data type, which can be inspected and manipulated reflectively.

DynamicData (p. 1461) allows manipulating a data sample whose type is unknown at compilation time. A **DynamicData** (p. 1461) object is created by passing its **DynamicType** (p. 1504), which can be a **StructType** (p. 2400), a **UnionType** (p. 2584), **ArrayType** (p. 828), **SequenceType** (p. 2344) or **AliasType** (p. 801). **DynamicData** (p. 1461) is a valid topic-type, so you can instantiate a Topic<DynamicData> (using the constructor that receives a **DynamicType** (p. 1504)), and then a DataReader<DynamicData> or a DataWriter<DynamicData>.

Several overloaded versions of the function **value()** (p. 1468) allow reading or writing the value of a member **by name or index** (p. 1465). Another set of functions, **loan_value()** (p. 1475), can access a non-primitive member (for reading and writing) without making a copy. For primitive arrays and sequences **DynamicData** (p. 1461) also provides a way to get or set all the values at once, using **set_values()** (p. 1474) and **get_values()** (p. 1471).

A number of other functions allow querying information about the sample and its type.

Applications that use both statically typed data and **DynamicData** (p. 1461) can benefit from the conversion between them, using rti::core::xtypes::convert().

For examples on how to use **DynamicData** (p. 1461), see **Using DynamicData** (p. 611).

8.98.2 Member Names and Indexes

As mentioned before, to access a member you need to specify its name or its index. Depending on the **DynamicType** (p. 1504) of the **DynamicData** (p. 1461) sample, the name may apply or not and the index can mean different things. This table summarizes that:

DynamicType (p. 1504) of the sample	Can access by name?	What does index mean?
StructType (p. 2400)	Yes	The position of the member in the type (starting at 1)
UnionType (p. 2584)	Yes	The value of the label that selects a member, see **get_discriminator()** (p. 1478)
ArrayType (p. 828)	No	The position of the element in the array (1 to **ArrayType::total_element_count()** (p. 832))
SequenceType (p. 2344)	No	The position of the element in the sequence (1 to **member_count()** (p. 1483)) or past **member_count()** (p. 1483) to increase the length of the sequence.
Any other type	N/A	N/A

For example, given the following IDL description of a type:

```
struct Foo {
    long x;
    long y;
};
```

If we have a **DynamicData** (p. 1461) `sample` representing data of that type, we can access the member `y` in two different ways:

```
sample.value("y", 33); // Set value to 33
sample.value(2, 33); // Same effect
```

Accessing a member by index can be more efficient, so when accessing a member repeatedly it may be beneficial to obtain its index first:

```
const uint32_t y_index = sample.member_index("y");
while (...) {
    int32_t y_value = sample.value<int32_t>(y_index);
    // ...
}
```

For an example on how to access a union member, see **get_discriminator()** (p. 1478), and for more examples, including how to access elements in a collection see **Using DynamicData** (p. 611).

8.98.2.1 Hierarchical Member Names

It is possible to refer to a nested member in a type without first having to use the **loan_value()** (p. 1475) API. You can do this by using a hierarchical name. A hierarchical member name is a concatenation of member names separated by the '.' character. The hierarchical name describes the complete path from a top-level type to the nested member. For example, in the following type:

```
struct MyNestedType {
    char theChar;
    octet theOctetArray[10];
    long long theMultidimensionalArray[4][6][12];
    sequence<long> myArrayOfSeq[8];
};
struct MyType {
    MyNestedType theNestedType;
```

```
};
```

any **DynamicData** (p. 1461) API that receives a member name will accept "theNestedType.theChar" to refer to the char member in MyNestedType:

```
char my_char = myDynamicData.value<char>("theNestedType.theChar");
```

In order to access the value of `theChar` without using a hierarchical name, you would have to first bind to theNested↩ Type and then get the value:

```
auto member = myDynamicData.loan_value("theNestedType");
char my_char = member.get().value<char>("theChar");
```

As you can see, using a hierarchical member name removes the need to call the **loan_value()** (p. 1475) API, and allows for access to nested members at any depth directly from the top-level type.

The member name can also contain indexes to address members in arrays and sequences. For example, to set the third member in the array `theOctetArray`, you can pass in "theNestedType.theOctetArray[2]" as the member name. The index values when used as part of the member name are 0-based.

For multi-dimensional arrays, the indexes for each dimension should be listed comma-separated in between brackets. For example, to address a member of `theMultidimensionalArray`, the member name should be something like "theNestedType.theMultidimensionalArray[3,2,5]".

In complex types with arrays and sequences that contain other arrays and sequences, the hierarchical name may include multiple index values, one right after another. For example, in MyNestedType, `myArrayOfSeq` is an array of sequences. In order to set the third member of the sequence in the fourth member of the array, the member name would be "myNestedType.myArrayOfSeq[3][2]".

8.98.3 Exceptions accessing a member

The different **DynamicData** (p. 1461) functions that access members may throw the following exceptions:

- **dds::core::InvalidArgumentError** (p. 1619) when attempting to **get** the value a member that doesn't exist in the type, is optional and is not set, or is part of a union and not currently selected (see get_discriminator). Attempting to **set** an unset optional member or an unselected union member is legal and will set the optional member or change the active union member.
- **dds::core::IllegalOperationError** (p. 1608) when attempting to get or set a member of the wrong type–that is, when the **DynamicType** (p. 1504) of that member doesn't match the template type `T` of the function. For example assumming that `my_string_member` is a string, the two following statements are illegal:

```
data.value("my_string_member", 3);
int v = data.value<int>("my_string_member");
```

8.98.3.1 Performance Considerations

By default, a **DynamicData** (p. 1461) object stores its content in an implementation-specific representation that allows getting and setting its fields by name or ID.

When a sample is received, the CDR-formatted data is deserialized into this buffer.

When a sample is published, the data is serialized into CDR from the buffer and sent to the network.

However, there are scenarios where **DynamicData** (p. 1461) is used for purposes that don't involve manipulating the object's content directly. Two common examples are:

- Recording/Replaying: Storing samples in persistent storage for later analysis or replay.
- Bridging: Transferring samples from one domain to another by transmitting samples received with a DynamicDataReader to another domain using a DynamicDataWriter.

RTI provides dedicated services, namely RTI Recording Service and RTI Routing Service, to address these use cases. However, in certain situations, you might be unable to utilize these services.

For scenarios where inspecting or modifying the content of **DynamicData** (p. 1461) fields is unnecessary, **DynamicData** (p. 1461) offers a high-performance method to configure a DataReader to avoid deserializing incoming samples into the internal representation. Additionally, the **DynamicData** (p. 1461) API provides a method to link a CDR-formatted serialization buffer directly with a **DynamicData** (p. 1461) object, facilitating direct transmission without converting the data from the internal representation to CDR.

To skip the deserialization of a **DynamicData** (p. 1461) object on the DataReader side, you can set the field DynamicDataTypeSerializationProperty_t::skip_deserialization to true.

Once a sample is received, you can access the serialized buffer directly by using the **dds::core::xtypes::DynamicData::get_cdr_buffer** (p. 1481). This buffer can be stored in persistent storage in a Recording/Replaying use case.

To link a serialization buffer directly with a **DynamicData** (p. 1461) object, you can use the **dds::core::xtypes::DynamicData::set_cdr_buffer** (p. 1482) method. This method allows you to set the buffer that will be used to serialize the **DynamicData** (p. 1461) object when it is published.

See also

Using DynamicData (p. 611)

8.98.4 Constructor & Destructor Documentation

8.98.4.1 DynamicData() [1/2]

```
dds::core::xtypes::DynamicData::DynamicData (
            const  dds::core::xtypes::DynamicType & type )
```

Creates a **DynamicData** (p. 1461) instance for a type.

Uses default DynamicDataProperty.

Precondition

!is_primitive(type) – a **DynamicData** (p. 1461) object cannot encapsulate a primitive type.

Parameters

type	The type of this data sample

8.98.4.2 DynamicData() [2/2]

```
dds::core::xtypes::DynamicData::DynamicData (
            const  dds::core::xtypes::DynamicType & type,
            const DynamicDataProperty & property )
```

Creates a **DynamicData** (p. 1461) instance for a type with specific memory-management.

Precondition

!is_primitive(type) – a **DynamicData** (p. 1461) object cannot encapsulate a primitive type.

Parameters

type	The type of this data sample
property	Memory-management configuration

8.98.5 Member Function Documentation

8.98.5.1 value() [1/4]

```
template<typename T >
void dds::core::xtypes::DynamicData::value (
            const std::string & name,
            const T & v )
```

Sets the value of a member by member name.

Template Parameters

T	The type of the member to set. The possible types are: • The types accepted by **PrimitiveType**<**T**> (p. 1959) • `std::string` (or **dds::core::string** (p. 435)), • **DynamicData** (p. 1461), when the member type is a nested struct or union. In this case you can use **loan_value()** (p. 1475) as well. • If the type of the member is an enumeration, use `int32_t`.

Note

If the member type is a collection type, use loan_value for collections of complex types or set_values for collections of primitive types.

For examples on how to set member values, see **Publishing data using DynamicData** (p. 611).

If the member is optional and unset, this function will set it. To unset it again, see **clear_optional_member()** (p. 1480).

It is not always possible to rely on inferring the type `T` based on the argument `v`. For example, if the member being set is a short, this statement will throw an exception, because `T` is deduced as `int`:

```
sample.value("my_short", 10);
```

There are two ways to specify that the member is a `short`:

```
sample.value<short>("my_short", 10);
sample.value("my_short", (short) 10);
```

Parameters

name	The member name, see **Member Names and Indexes** (p. 1465). This function also allows to access nested members
v	The value to set for that member

Exceptions

See	**Exceptions accessing a member** (p. 1466)

See also

member_exists(const std::string&) const (p. 1483)

Publishing data using DynamicData (p. 611)

8.98.5.2 value() [2/4]

```
template<typename T >
void dds::core::xtypes::DynamicData::value (
            uint32_t index,
            const T & v )
```

Sets the value of a member by member index.

Parameters

index	The position of the member in the type, see **Member Names and Indexes** (p. 1465).
v	The value to set for that member

Exceptions

See	**Exceptions accessing a member** (p. 1466)

See also

value(const std::string&, const T&) (p. 1468)

Member Names and Indexes (p. 1465)

8.98.5.3 value() [3/4]

```
template<typename T >
T dds::core::xtypes::DynamicData::value (
            const std::string & name ) const
```

Gets the value of a member by member name.

Template Parameters

T	The type of the member. See **value(const std::string&, const T&)** (p. 1468)

For examples on how to get member values, see **Subscribing to data using DynamicData** (p. 612).

Parameters

name	The member name, see **Member Names and Indexes** (p. 1465).

Returns

The value of that member

Exceptions

See	**Exceptions accessing a member** (p. 1466)

See also

value(const std::string&, const T&) (p. 1468)

Subscribing to data using DynamicData (p. 612)

8.98.5.4 value() [4/4]

```
template<typename T >
T dds::core::xtypes::DynamicData::value (
            uint32_t index ) const
```

Gets the value of a member by member index.

Template Parameters

T	The type of the member. See **value(const std::string&, const T&)** (p. 1468)

For examples on how to get member values, see **Subscribing to data using DynamicData** (p. 612).

Parameters

index	The position of the member in the type, see **Member Names and Indexes** (p. 1465).

Returns

The value of that member

Exceptions

See	**Exceptions accessing a member** (p. 1466)

See also

value(const std::string&, const T&) (p. 1468)

Member Names and Indexes (p. 1465)

8.98.5.5 get_values() [1/4]

```
template<typename T >
void dds::core::xtypes::DynamicData::get_values (
            const std::string & name,
            std::vector< T > & out_array ) const
```

Obtains the values of an array or sequence member by member name.

This function reuses an existing vector

Template Parameters

T	The element type of the array or sequence. These are the possible types: • The primitive types `char`, `(u)int8_t`, `(u)int16_t`, `DDS_Long`, `DDS_UnsignedLong`, `DDS_LongLong`, `DDS_UnsignedLongLong`, `float`, `double`. The types `int` and `unsigned int` may be used instead of `DDS_Long` and `DDS_UnsignedLong` except in some platforms (VxWorks). • If the element type is an enumeration, use `int32_t`. • If the element type is `bool`, use `uint8_t`.

Parameters

name	The member name, see **Member Names and Indexes** (p. 1465).
out_array	A vector containing the values of that array or sequence. The vector will be resized accordingly.

8.98.5.6 get_values() [2/4]

```
template<typename T >
void dds::core::xtypes::DynamicData::get_values (
            uint32_t index,
            std::vector< T > & out_array ) const
```

Obtains the values of an array or sequence member by member index.

This function reuses an existing vector

Template Parameters

T	The element type of the array or sequence. (See **get_values(const std::string&, std::vector<T>&) const** (p. 1471))

Parameters

index	The position of the array or sequence member in the type, see **Member Names and Indexes** (p. 1465).
out_array	A vector containing the values of that array or sequence. The vector will be resized accordingly.

See also

Member Names and Indexes (p. 1465)

get_values(const std::string& name, std::vector<T>& out_array) const (p. 1471)

8.98.5.7 get_values() [3/4]

```
template<typename T >
std::vector< T > dds::core::xtypes::DynamicData::get_values (
            uint32_t index ) const
```

Obtains the values of an array or sequence member by member index.

This function returns a new vector.

Template Parameters

T	The element type of the array or sequence. (See **get_values(const std::string&, std::vector<T>&) const** (p. 1471))

Parameters

index	The position of the array or sequence member in the type

Returns

A vector containing the values of that array or sequence.

See also

Member Names and Indexes (p. 1465)

get_values(const std::string& name, std::vector<T>& out_array) const (p. 1471)

8.98.5.8 get_values() [4/4]

```
template<typename T >
std::vector< T > dds::core::xtypes::DynamicData::get_values (
            const std::string & name ) const
```

Obtains the values of an array or sequence member by member name.

This function returns a new vector.

Template Parameters

T	The element type of the array or sequence. (See **get_values(const std::string&, std::vector<T>&) const** (p. 1471))

Parameters

name	The member name, see **Member Names and Indexes** (p. 1465).

Returns

A vector containing the values of that array or sequence.

See also

get_values(const std::string& name, std::vector<T>& out_array) const (p. 1471)

8.98.5.9 set_values() [1/2]

```
template<typename T >
void dds::core::xtypes::DynamicData::set_values (
            uint32_t index,
            const std::vector< T > & v )
```

Sets the values of an array or sequence member by member index.

Template Parameters

T	The element type of the array or sequence. (See **get_values(const std::string&, std::vector<T>&) const** (p. 1471))

Parameters

index	The position of the array or sequence member in the type
v	A vector containing the values to be set for that array or sequence.

See also

Member Names and Indexes (p. 1465)

get_values(const std::string& name, std::vector<T>& out_array) const (p. 1471)

8.98.5.10 set_values() [2/2]

```
template<typename T >
void dds::core::xtypes::DynamicData::set_values (
```

```
            const std::string & name,
            const std::vector< T > & v )
```

Sets the values of an array or sequence member by member name.

Template Parameters

T	The element type of the array or sequence. (See **get_values(const std::string&, std::vector<T>&) const** (p. 1471))

Parameters

name	The member name, see **Member Names and Indexes** (p. 1465).
v	A vector containing the values to be set for that array or sequence.

See also

get_values(const std::string& name, std::vector<T>& out_array) const (p. 1471)

8.98.5.11 loan_value() [1/4]

```
LoanedDynamicData dds::core::xtypes::DynamicData::loan_value (
            const std::string & name )
```

Gets a view of a complex member.

This function returns a new LoanedDynamicData instance that provides a view of a complex member (a struct or a union).

Only a single member of a given **DynamicData** (p. 1461) object may be loaned at a time. However, members of members may be recursively loaned to any depth. Furthermore, while the outer object has a bound member, it may only be modified through that bound member. That is, after loaning this member, the outer object disables all calls to set **value()** (p. 1468) until the loan is returned.

This method, as opposed to **value<DynamicData>()** (p. 1468) doesn't make a copy; it provides a view of the member so changes to the loaned member change the outer object.

Parameters

name	The member name, see **Member Names and Indexes** (p. 1465).

Returns

The object to access the complex member

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::core::xtypes::LoanedDynamicData (p. 1656)

Using DynamicData (p. 611).

8.98.5.12 loan_value() [2/4]

```
LoanedDynamicData dds::core::xtypes::DynamicData::loan_value (
            uint32_t index )
```

Gets a view of a complex member.

This function returns a new LoanedDynamicData instance that provides a view of a complex member (a struct or a union).

Parameters

index	The position of the member in the type, see **Member Names and Indexes** (p. 1465).

Returns

The object to access the complex member

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

loan_value(const std::string&) (p. 1475)

rti::core::xtypes::LoanedDynamicData (p. 1656)

Member Names and Indexes (p. 1465)

8.98.5.13 loan_value() [3/4]

```
LoanedDynamicData & dds::core::xtypes::DynamicData::loan_value (
            LoanedDynamicData & data,
            const std::string & name )
```

Gets a view of a complex member.

This function reuses a LoanedDynamicData instance, returning its previous loan.

Parameters

data	The object to access the complex member
name	The member name, see **Member Names and Indexes** (p. 1465).

See also

loan_value(const std::string&) (p. 1475)

rti::core::xtypes::LoanedDynamicData (p. 1656)

8.98.5.14 loan_value() [4/4]

```
LoanedDynamicData & dds::core::xtypes::DynamicData::loan_value (
            LoanedDynamicData & data,
            uint32_t mid )
```

Gets a view of a complex member.

This function reuses a LoanedDynamicData instance, returning its previous loan.

Parameters

data	The object to access the complex member
mid	The position of the member in the type, see **Member Names and Indexes** (p. 1465).

See also

loan_value(const std::string&) (p. 1475)

rti::core::xtypes::LoanedDynamicData (p. 1656)

8.98.5.15 discriminator_value()

```
uint32_t dds::core::xtypes::DynamicData::discriminator_value ( ) const
```

Deprecated. Use **get_discriminator()** (p. 1478) instead. Obtains the value of the union discriminator (**UnionType** (p. 2584) only)

Use the value that this function returns in any function expecting a member index to access the active member in this union.

For example, given the following IDL Union:
```
union Foo switch (short) {
case 1:
    long x;
case 2:
case 3:
    string y;
};
```

If we have a **DynamicData** (p. 1461) `sample` of that type whose active member is "y", we can access it as follows:
```
std::string y_value = sample.value<std::string>(sample.discriminator_value());
y_value = sample.value<std::string>("y"); // Same effect
y_value = sample.value<std::string>(3); // Same effect
```

If you set a different member, the discriminator value automatically changes:
```
sample.value("x", 10);
assert (sample.discriminator_value() == 1);
```

Precondition

type() (p. 1482).kind() == TypeKind::UNION_TYPE – Only applies to samples of a **UnionType** (p. 2584).

Returns

The value of this sample's discriminator.

8.98.5.16 get_discriminator()

```
int32_t dds::core::xtypes::DynamicData::get_discriminator ( ) const
```

Obtains the value of the union discriminator (**UnionType** (p. 2584) only)

Use the value that this function returns in any function expecting a member index to access the active member in this union.

For example, given the following IDL Union:
```
union Foo switch (short) {
case 1:
    long x;
case 2:
case 3:
    string y;
};
```

If we have a **DynamicData** (p. 1461) `sample` of that type whose active member is "y", we can access it as follows:
```
std::string y_value = sample.value<std::string>(sample.get_discriminator());
y_value = sample.value<std::string>("y"); // Same effect
y_value = sample.value<std::string>(3); // Same effect
```

If you set a different member, the discriminator value automatically changes:
```
sample.value("x", 10);
assert (sample.get_discriminator() == 1);
```

Precondition

type() (p. 1482).kind() == TypeKind::UNION_TYPE – Only applies to samples of a **UnionType** (p. 2584).

Returns

The value of this sample's discriminator.

8.98.5.17 set_discriminator()

```
void dds::core::xtypes::DynamicData::set_discriminator (
            int32_t disc )
```

Sets the value of the union discriminator (**UnionType** (p. 2584) only)

If the discriminator value identifies the same member in the union, the previously set value for the member is preserved and only the discriminator value is updated. If the discriminator value identifies a different member in the union, the previously set member in the union will be finalized and the identified member will be initialized to its default value. If the discriminator doesn't identify any member in the union, then the previously set member will be finalized and the discriminator will be set to the provided value.

For example, given the following IDL Union:

```
union Foo switch (short) {
case 1:
    long x;
case 2:
case 3:
    string y;
};
```

We can set the active member to "y" as follows:

```
// Set the discriminator to 2 and select member "y"
sample.set_discriminator(2);
assert (sample.get_discriminator() == 2);
y_value = sample.value<std::string>("y");
y_value = sample.value<std::string>(2);
// Set "y" to a value
sample.value<std::string>(2, "hello")
```

Setting the discriminator to 3 also selects "y" and preserves the previous value.

```
sample.set_discriminator(3);
assert (sample.get_discriminator() == 3);
assert (sample.value<std::string>("y") == "hello");
```

Setting the discriminator to a new value that selects a different member, "x", finalizes the previous member's value and initializes the new member to its default.

```
sample.set_discriminator(1);
assert (sample.value<int32_t>("x") == 0);
y_value = sample.value<std::string>("y"); // throws an exception
```

The discriminator can also be set to a value that does not select any member.

```
sample.set_discriminator(100);
assert (sample.get_discriminator() == 100);
assert (sample.member_count() == 0);
```

Precondition

type() (p. 1482).kind() == TypeKind::UNION_TYPE – Only applies to samples of a **UnionType** (p. 2584).

8.98.5.18 clear_all_members()

```
void dds::core::xtypes::DynamicData::clear_all_members ( )
```

Clear the contents of all data members of this object, including key members.

MT Safety:

UNSAFE.

Exceptions

One	of the **Standard Exceptions** (p. 428)

This API is not supported when the **DynamicData** (p. 1461) object is in CDR format and will fail with **dds::core::PreconditionNotMetError** (p. 1941). See **dds::core::xtypes::DynamicData::is_cdr** (p. 1481) for more information.

See also

dds::core::xtypes::DynamicData::clear_optional_member (p. 1480)

8.98.5.19 clear_optional_member() [1/2]

```
void dds::core::xtypes::DynamicData::clear_optional_member (
            const std::string & name )
```

Clear the contents of a single optional data member of this object.

Parameters

name	The member name, see **Member Names and Indexes** (p. 1465).

8.98.5.20 clear_optional_member() [2/2]

```
void dds::core::xtypes::DynamicData::clear_optional_member (
            uint32_t index )
```

Clear the contents of a single optional data member of this object.

Parameters

index	The position of the member in the type, see **Member Names and Indexes** (p. 1465).

8.98.5.21 clear_member() [1/2]

```
void dds::core::xtypes::DynamicData::clear_member (
            const std::string & name )
```

Clear the contents of a single data member of this object.

Parameters

name	The member name, see **Member Names and Indexes** (p. 1465).

8.98.5.22 clear_member() [2/2]

```
void dds::core::xtypes::DynamicData::clear_member (
            uint32_t index )
```

Clear the contents of a single data member of this object.

Parameters

index	The position of the member in the type, see **Member Names and Indexes** (p. 1465).

8.98.5.23 is_cdr()

```
bool dds::core::xtypes::DynamicData::is_cdr ( ) const
```

Determines if this **DynamicData** (p. 1461) sample is represented as CDR.

This method returns true if the **DynamicData** (p. 1461) object is using CDR as its data representation, and false otherwise.

When the **DynamicData** (p. 1461) object is using CDR as its data representation, many of the **DynamicData** (p. 1461) APIs cannot be used. The **DynamicData** (p. 1461) object will use CDR as its data representation when DynamicData↩ TypeSerializationProperty_t::skip_deserialization has been set to true or **dds::core::xtypes::DynamicData::set_cdr↩ _buffer** (p. 1482) has been used to associate a CDR buffer with the **DynamicData** (p. 1461) object. See each functions's documentation for whether or not the function is supported for **DynamicData** (p. 1461) objects in CDR format.

See also

dds::core::xtypes::DynamicData::get_cdr_buffer (p. 1481)

dds::core::xtypes::DynamicData::set_cdr_buffer (p. 1482)

DynamicDataTypeSerializationProperty_t::skip_deserialization

8.98.5.24 get_cdr_buffer()

```
std::pair< const char *, uint32_t > dds::core::xtypes::DynamicData::get_cdr_buffer ( ) const
```

Gets the CDR buffer of this **DynamicData** (p. 1461) sample.

This method returns the CDR buffer associated with a **DynamicData** (p. 1461) object.

If the **DynamicData** (p. 1461) object is not using CDR as its data representation, this method will throw an exception.

To configure a **DynamicData** (p. 1461) DataReader to not deserialize the incoming samples and keep the data in CDR format, you must set the configuration option DynamicDataTypeSerializationProperty_t::skip_deserialization to true.

MT Safety:

UNSAFE.

Parameters

length	<<***out***>> *(p. 355)* CDR buffer length.

Exceptions

One	of the **Standard Exceptions** (p. 428)

Returns

The buffer

8.98.5.25 set_cdr_buffer()

```
void dds::core::xtypes::DynamicData::set_cdr_buffer (
            const char * buffer,
            uint32_t length )
```

Sets the CDR buffer of this **DynamicData** (p. 1461) sample.

Parameters

buffer	The buffer to set
length	The size of the buffer

8.98.5.26 type()

```
const  dds::core::xtypes::DynamicType & dds::core::xtypes::DynamicData::type ( ) const
```

Gets the data type of this **DynamicData** (p. 1461).

8.98.5.27 type_kind()

```
 dds::core::xtypes::TypeKind dds::core::xtypes::DynamicData::type_kind ( ) const
```

Gets the data type kind of this **DynamicData** (p. 1461).

Returns

this->**type()** (p. 1482).kind().

8.98.5.28 member_count()

```
uint32_t dds::core::xtypes::DynamicData::member_count ( ) const
```

Get the number of members in this sample.

For objects of type **ArrayType** (p. 828) or **SequenceType** (p. 2344), this method returns the number of elements in the collection.

For objects of type kind **StructType** (p. 2400), it returns the number of fields in the sample, which will always be the same as the number of fields in the type.

8.98.5.29 member_exists() [1/2]

```
bool dds::core::xtypes::DynamicData::member_exists (
             const std::string & name ) const
```

Indicates whether a member exists in this sample.

If the member doesn't exist in the type, this function returns false. In all other cases, it provides the same result as **rti::core::xtypes::DynamicDataMemberInfo::member_exists()** (p. 1497), which is retrieved with **member_info()** (p. 1484)).

Parameters

name	The member name, see **Member Names and Indexes** (p. 1465).

8.98.5.30 member_exists() [2/2]

```
bool dds::core::xtypes::DynamicData::member_exists (
            uint32_t index ) const
```

Indicates whether a member exists in this sample.

If the member doesn't exist in the type, this function returns false. In all other cases, it provides the same result as **rti::core::xtypes::DynamicDataMemberInfo::member_exists()** (p. 1497), which is retrieved with **member_info()** (p. 1484)).

Parameters

index	The member index, see **Member Names and Indexes** (p. 1465).

8.98.5.31 member_exists_in_type() [1/2]

```
bool dds::core::xtypes::DynamicData::member_exists_in_type (
            const std::string & name ) const
```

Indicates whether a member with a particular name exists in **type()** (p. 1482)

8.98.5.32 member_exists_in_type() [2/2]

```
bool dds::core::xtypes::DynamicData::member_exists_in_type (
            uint32_t index ) const
```

Indicates whether a member of a particular index exists in **type()** (p. 1482)

8.98.5.33 info()

```
DynamicDataInfo dds::core::xtypes::DynamicData::info ( ) const
```

Returns information about this sample.

8.98.5.34 member_info() [1/2]

```
rti::core::xtypes::DynamicDataMemberInfo dds::core::xtypes::DynamicData::member_info (
            const std::string & name ) const
```

Returns information about a member.

Precondition

this->**type_kind()** (p. 1483) must be **dds::core::xtypes::TypeKind_def::ARRAY_TYPE** (p. 2573), **dds::core::xtypes::TypeKind_def::SEQUENCE_TYPE** (p. 2573), **dds::core::xtypes::TypeKind_def::STRUCTURE_TYPE** (p. 2574), or **dds::core::xtypes::TypeKind_def::UNION_TYPE** (p. 2574)

When `name` identifies a field in a struct or union:

- If the specified member is not defined in the type, this function throws **dds::core::InvalidArgumentError** (p. 1619).
- If the specified member is defined in the type but doesn't exist in this data sample, this function returns an object with **rti::core::xtypes::DynamicDataMemberInfo::member_exists()** (p. 1497) set to false.
- If the specified member is defined in the type and exists in this data sample, **rti::core::xtypes::DynamicDataMemberInfo::member_exists** (p. 1497) is true.

When `name` identifies a sequence element, such as `"my_sequence[i]"`:

- If `i` is greater than the sequence's maximum length, this function throws **dds::core::InvalidArgumentError** (p. 1619).
- If `i` is greater than the sequence's current length but smaller than its maximum length, this function returns an object with **rti::core::xtypes::DynamicDataMemberInfo::member_exists()** (p. 1497) set to false.
- If `i` is smaller than or equal to the current length, **rti::core::xtypes::DynamicDataMemberInfo::member_exists()** (p. 1497) is true.

When `name` identifies an array element, such as `"my_array[i]`, this function either throws when the `i` is out of bounds or else returns an object with **rti::core::xtypes::DynamicDataMemberInfo::member_exists()** (p. 1497) set to true.

The **rti::core::xtypes::DynamicDataMemberInfo** (p. 1495) returned will contain information about the identified member, including its type, whether it exists in this sample, and other metadata.

The bounds for strings and collections are not provided in the returned **rti::core::xtypes::DynamicDataMemberInfo** (p. 1495). To access that information you must query the **dds::core::xtypes::DynamicType** (p. 1504) of the member's type.

The **dds::core::xtypes::DynamicType** (p. 1504) of a **DynamicData** (p. 1461) object can be accessed using **dds::core::xtypes::DynamicData::type()** (p. 1482). You must iterate through the **dds::core::xtypes::DynamicType** (p. 1504) to locate the members **dds::core::xtypes::DynamicType** (p. 1504). Once obtained, you can query the bounds using the following APIs:

- For strings and sequences: **dds::core::xtypes::UnidimensionalCollectionType::bounds()** (p. 2577)
- For arrays: **dds::core::xtypes::ArrayType::dimension_count()** (p. 832) and **dds::core::xtypes::ArrayType::dimension()** (p. 832)

Parameters

name	The member name, see **Member Names and Indexes** (p. 1465).

Returns

Information about the member.

Exceptions

dds::core::InvalidArgumentError *(p. 1619)*	if the member doesn't exist in the type, or **Standard Exceptions** (p. 428).

8.98.5.35 member_info() [2/2]

```
rti::core::xtypes::DynamicDataMemberInfo dds::core::xtypes::DynamicData::member_info (
            uint32_t index ) const
```

Returns information about a member.

Precondition

this->**type_kind()** (p. 1483) must be **dds::core::xtypes::TypeKind_def::ARRAY_TYPE** (p. 2573), **dds::core::xtypes::TypeKind_def::SEQUENCE_TYPE** (p. 2573), or **dds::core::xtypes::TypeKind_def::STRUCTURE_TYPE** (p. 2574), **dds::core::xtypes::TypeKind_def::UNION_TYPE** (p. 2574)

When this sample represents a struct and `index` is the position of the member in the type definition, or when the sample represents a union and `index` is the discriminator value:

- If the specified member is not defined in the type, this function throws **dds::core::InvalidArgumentError** (p. 1619).
- If the specified member is defined in the type but doesn't exist in this data sample, this function returns an object with **rti::core::xtypes::DynamicDataMemberInfo::member_exists()** (p. 1497) set to false.
- If the specified member is defined in the type and exists in this data sample, **rti::core::xtypes::DynamicDataMemberInfo::member_exists** (p. 1497) is true.

When this sample represents a sequence and `index` is the 1-based element index:

- If `index` is greater than the sequence's maximum length, this function throws **dds::core::InvalidArgumentError** (p. 1619).
- If `index` is greater than the sequence's current length but smaller than its maximum length, this function returns an object with **rti::core::xtypes::DynamicDataMemberInfo::member_exists()** (p. 1497) set to false.

- If `index` is smaller than or equal to the current length, **rti::core::xtypes::DynamicDataMemberInfo::member_exists()** (p. 1497) is true.

When this sample represents an array this function either throws **dds::core::InvalidArgumentError** (p. 1619) when the index is out of bounds or else returns an object with **rti::core::xtypes::DynamicDataMemberInfo::member_exists()** (p. 1497) set to true.

The **rti::core::xtypes::DynamicDataMemberInfo** (p. 1495) returned will contain information about the identified member, including its type, whether it exists in this sample, and other metadata.

The bounds for strings and collections are not provided in the returned **rti::core::xtypes::DynamicDataMemberInfo** (p. 1495). To access that information you must query the **dds::core::xtypes::DynamicType** (p. 1504) of the member's type.

The **dds::core::xtypes::DynamicType** (p. 1504) of a **DynamicData** (p. 1461) object can be accessed using **dds::core::xtypes::DynamicData::type()** (p. 1482). You must iterate through the **dds::core::xtypes::DynamicType** (p. 1504) to locate the members **dds::core::xtypes::DynamicType** (p. 1504). Once obtained, you can query the bounds using the following APIs:

- For strings and sequences: **dds::core::xtypes::UnidimensionalCollectionType::bounds()** (p. 2577)
- For arrays: **dds::core::xtypes::ArrayType::dimension_count()** (p. 832) and **dds::core::xtypes::ArrayType::dimension()** (p. 832)

Parameters

index	The member index, see **Member Names and Indexes** (p. 1465).

Returns

DynamicDataMemberInfo Information about the member.

Exceptions

dds::core::InvalidArgumentError *(p. 1619)*	if the member doesn't exist in the type or the element with this index doesn't exist in this sequence. See also **Standard Exceptions** (p. 428).

8.98.5.36 member_index()

```
uint32_t dds::core::xtypes::DynamicData::member_index (
            const std::string & name ) const
```

Translates from member name to member index.

Parameters

name	The member name, see **Member Names and Indexes** (p. 1465).

Returns

The member index for the member with that name

Exceptions

See	**Exceptions accessing a member** (p. 1466)

8.98.5.37 is_member_key() [1/2]

```
bool dds::core::xtypes::DynamicData::is_member_key (
            const std::string & name ) const
```

Returns whether a member is a key.

Parameters

name	The member name, see **Member Names and Indexes** (p. 1465).

Exceptions

See	**Exceptions accessing a member** (p. 1466)

8.98.5.38 is_member_key() [2/2]

```
bool dds::core::xtypes::DynamicData::is_member_key (
            uint32_t index ) const
```

Returns whether a member is a key.

Parameters

index	The member index, see **Member Names and Indexes** (p. 1465).

Exceptions

See	**Exceptions accessing a member** (p. 1466)

8.98.5.39 member_type() [1/2]

```
dds::core::xtypes::DynamicType dds::core::xtypes::DynamicData::member_type (
            const std::string & name ) const
```

Returns the type of a member.

8.98.5.40 member_type() [2/2]

```
dds::core::xtypes::DynamicType dds::core::xtypes::DynamicData::member_type (
            uint32_t index ) const
```

Returns the type of a member.

8.98.5.41 from_string()

```
void dds::core::xtypes::DynamicData::from_string (
            RTI_STRING_IN str,
             rti::topic::PrintFormatKind format_kind = rti::topic::PrintFormatKind::JSON )  [inline]
```

Populates this **DynamicData** (p. 1461) object from a JSON string representation.

This method takes a JSON string representing a data sample of this type, and uses it to populate this **DynamicData** (p. 1461).

Parameters

str	<<***in***>> *(p. 355).* The JSON string representation
format_kind	<<***in***>> *(p. 355).* Must be set to PrintFormatKind::JSON_PRINT_FORMAT, the only currently supported format.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::OutOfResourcesError** (p. 1898)

Referenced by **rti::topic::from_string()**.

8.98.6 Friends And Related Function Documentation

8.98.6.1 to_cdr_buffer()

```
std::vector< char > & to_cdr_buffer (
            std::vector< char > & buffer,
            const  DynamicData & sample,
             dds::core::policy::DataRepresentationId representation_id =  dds::core::policy::↩
DataRepresentation::auto_id() )  [related]
```

<<***extension***>> *(*p. *354)* Serializes a **DynamicData** (p. 1461) sample into CDR format

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

Parameters

buffer	The output buffer, which will be resized to the correct size
sample	The sample to serialize
representation↩ _id	The data representation used to serialize the data

Returns

A reference to `buffer`

Examples

Foo.hpp.

8.98.6.2 from_cdr_buffer()

```
DynamicData & from_cdr_buffer (
             DynamicData & sample,
            const std::vector< char > & buffer )  [related]
```

<<***extension***>> *(*p. *354)* Creates a **DynamicData** (p. 1461) sample by deserializing a CDR buffer'

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

Parameters

sample	The sample to deserialize.
buffer	The buffer containing the CDR data

Returns

A reference to `sample`

Examples

Foo.hpp.

8.98.6.3 convert() [1/2]

```
template<typename TopicType >
TopicType convert (
            const  DynamicData & sample )  [related]
```

<<***extension***>> *(p. 354)* Creates a typed sample from a **DynamicData** (p. 1461) sample.

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

This function is useful when an application uses **DynamicData** (p. 1461) and regular typed Topics at the same time.

For example, given this IDL type:

```
struct Foo {
    long x;
};
```

We can convert from **Foo** (p. 1588) and **DynamicData** (p. 1461):

```
using dds::core::xtypes::DynamicData;
// Receive a DynamicData sample
dds::sub::DataReader<DynamicData> reader = // ...;
dds::sub::LoanedSamples<DynamicData> samples = reader.take();
assert (samples.size() > 0);
// Create a Foo sample
const DynamicData& dynamic_data = samples[0].data();
assert (rti::core::xtypes::can_convert<Foo>(dynamic_data));
Foo typed_data = rti::core::xtypes::convert<Foo>(dynamic_data);
assert (typed_data.x() == dynamic_data.value<int32_t>("x"));
```

Precondition

can_convert<TopicType>(sample), otherwise the behavior is undefined.

Template Parameters

TopicType	An IDL topic-type

Parameters

sample	The **DynamicData** (p. 1461) sample to convert to type `TopicType`

Returns

An instance of `TopicType` containing the data in `sample`.

8.98.6.4 convert() [2/2]

```
template<typename TopicType >
DynamicData convert (
            const TopicType & sample )  [related]
```

<<***extension***>> *(p. 354)* Creates a **DynamicData** (p. 1461) sample from a typed sample

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

Template Parameters

TopicType	An IDL topic-type

Parameters

sample	The typed sample to convert to **DynamicData** (p. 1461)

Returns

A new **DynamicData** (p. 1461) object whose type is rti::topic::dynamic_type<TopicType> and contains the data from `sample`

8.98.6.5 can_convert()

```
template<typename TopicType >
bool can_convert (
            const  DynamicData & sample )  [related]
```

<<***extension***>> *(p. 354)* Determines if the **DynamicType** (p. 1504) of this sample is equal to the **DynamicType** (p. 1504) of `TopicType`

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

Template Parameters

TopicType	An IDL topic-type

Returns

True if the **DynamicType** (p. 1504) of TopicType is the same as this sample's and therefore it is possible to create an instance of `TopicType` from this **DynamicData** (p. 1461) object.

See also

convert() (p. 1491)

8.98.6.6 get_tuple()

```
template<typename...  Types>
std::tuple< Types...  > get_tuple (
            const  dds::core::xtypes::DynamicData & data )  [related]
```

<<***C++11***>> *(p. 353)* <<***experimental***>> *(p. 355)* <<***extension***>> *(p. 354)* Retrieves all the values of a **DynamicData** (p. 1461) object into an std::tuple at once

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

See also

Manipulating data reflectively using C++ tuples (p. 612)

8.98.6.7 set_tuple()

```
template<typename... Types>
void set_tuple (
             dds::core::xtypes::DynamicData & data,
            const std::tuple< Types... > & value ) [related]
```

<<***C++11***>> (p. *353)* <<***experimental***>> (p. *355)* <<***extension***>> (p. *354)* Assigns the values of a std::tuple to a **DynamicData** (p. 1461) object at once

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

See also

Manipulating data reflectively using C++ tuples (p. 612)

8.99 rti::core::xtypes::DynamicDataInfo Class Reference

Contains information about a DynamicData sample.

```
#include <DynamicDataInfo.hpp>
```

Public Member Functions

- int32_t **member_count** () const

  The number of members of this sample.
- int32_t **stored_size** () const

  The number of bytes currently used to store the data of this sample.

8.99.1 Detailed Description

Contains information about a DynamicData sample.

See also

DynamicData::info()

8.99.2 Member Function Documentation

8.99.2.1 member_count()

```
int32_t rti::core::xtypes::DynamicDataInfo::member_count ( ) const
```

The number of members of this sample.

8.99.2.2 stored_size()

```
int32_t rti::core::xtypes::DynamicDataInfo::stored_size ( ) const
```

The number of bytes currently used to store the data of this sample.

8.100 rti::core::xtypes::DynamicDataMemberInfo Class Reference

Contains information about a DynamicData member.

```
#include <DynamicDataMemberInfo.hpp>
```

Public Member Functions

- uint32_t **member_index** () const
  The member index.
- const **dds::core::string** & **member_name** () const
  The member name.
- **dds::core::xtypes::TypeKind** **member_kind** () const
  The type kind of this member.
- uint32_t **element_count** () const
  The number of elements in this member.
- **dds::core::xtypes::TypeKind** **element_kind** () const
  If this member is a collection, this function returns the type kind of its elements.
- bool **member_exists** () const
  Indicates if this member is present in this sample.

8.100.1 Detailed Description

Contains information about a DynamicData member.

See also

dds::core::xtypes::DynamicData::member_info() (p. 1484)

8.100.2 Member Function Documentation

8.100.2.1 member_index()

```
uint32_t rti::core::xtypes::DynamicDataMemberInfo::member_index ( ) const
```

The member index.

See also

Member Names and Indexes (p. 1465).

8.100.2.2 member_name()

```
const  dds::core::string & rti::core::xtypes::DynamicDataMemberInfo::member_name ( ) const
```

The member name.

See also

Member Names and Indexes (p. 1465).

8.100.2.3 member_kind()

```
dds::core::xtypes::TypeKind rti::core::xtypes::DynamicDataMemberInfo::member_kind ( ) const
```

The type kind of this member.

This a convenience function, equivalent to looking up the member's DynamicType::kind().

See also

dds::core::xtypes::DynamicData::type() (p. 1482)

dds::core::xtypes::DynamicType::kind() (p. 1506)

8.100.2.4 element_count()

```
uint32_t rti::core::xtypes::DynamicDataMemberInfo::element_count ( ) const
```

The number of elements in this member.

Only applies to members of ArrayType and SequenceType. For any other type this is always zero.

8.100.2.5 element_kind()

```
dds::core::xtypes::TypeKind rti::core::xtypes::DynamicDataMemberInfo::element_kind ( ) const
```

If this member is a collection, this function returns the type kind of its elements.

This function provides the same information as **dds::core::xtypes::CollectionType::content_type()** (p. 939).

See also

dds::core::xtypes::DynamicData::type() (p. 1482)
dds::core::xtypes::CollectionType::content_type() (p. 939)

8.100.2.6 member_exists()

```
bool rti::core::xtypes::DynamicDataMemberInfo::member_exists ( ) const
```

Indicates if this member is present in this sample.

A member that is defined in a type may not exist in a sample of that type in the following situations

- The member is optional and is unset
- Being part of a union, the discriminator doesn't select this member
- When getting the information about a sequence element, with index `i`, and `i` is greater than the current sequence length but smaller than its maximum length.

See also

dds::core::xtypes::DynamicData::member_exists_in_type (p. 1484)

8.101 rti::core::xtypes::DynamicDataProperty Class Reference

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Specifies the properties of a DynamicData object

```
#include <rti/core/xtypes/DynamicDataProperty.hpp>
```

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- int32_t **buffer_initial_size** () const

  Getter (see setter with the same name)
- **DynamicDataProperty** & **buffer_initial_size** (int32_t value)

  Sets the initial size of the buffer.
- int32_t **buffer_max_size** () const

  Getter (see setter with the same name)
- **DynamicDataProperty** & **buffer_max_size** (int32_t value)

  Sets the maximum size of the buffer.

8.101.1 Detailed Description

*<<**extension**>>* (p. *354*) *<<**value-type**>>* (p. *350*) Specifies the properties of a DynamicData object

8.101.2 Member Function Documentation

8.101.2.1 buffer_initial_size() [1/2]

```
int32_t rti::core::xtypes::DynamicDataProperty::buffer_initial_size ( ) const  [inline]
```

Getter (see setter with the same name)

8.101.2.2 buffer_initial_size() [2/2]

```
DynamicDataProperty & rti::core::xtypes::DynamicDataProperty::buffer_initial_size (
            int32_t value )  [inline]
```

Sets the initial size of the buffer.

Parameters

value	The new initial size of the buffer

This property is used to configure the DynamicData objects that are created stand-alone as well as the DynamicData samples that are obtained from the sample pool that is created by each DynamicData DataReader.

If set to 0 (default): The initial buffer size will be set to the minimum amount of space required to hold the overhead required by the DynamicData internal representation (about 100 bytes) in addition to the minimum deserialized size of

a sample. The minimum deserialized size of a sample assumes that all strings are allocated to their default values, sequences are left to length 0, and all optional members are unset.

If set to any value other than 0: The underlying buffer will be allocated to the provided size plus the overhead required by the DynamicData internal representation (about 100 bytes). If the provided size plus the overhead is less than the size used when buffer_initial_size is left to 0, then the default value is used.

See also

rti::core::xtypes::DynamicDataProperty::buffer_max_size (p. 1499)

8.101.2.3 buffer_max_size() [1/2]

```
int32_t rti::core::xtypes::DynamicDataProperty::buffer_max_size ( ) const  [inline]
```

Getter (see setter with the same name)

8.101.2.4 buffer_max_size() [2/2]

```
DynamicDataProperty & rti::core::xtypes::DynamicDataProperty::buffer_max_size (
            int32_t value )  [inline]
```

Sets the maximum size of the buffer.

Parameters

value	The new maximum size of the buffer

This property is used to configure the DynamicData objects that are created stand-alone as well as the DynamicData samples that are obtained from the sample pool that is created by each DynamicData DataReader.

A DynamicData object will grow to this size from the initial size as needed. The buffer_max_size includes all overhead that is required for the internal DynamicData representation and therefore represents a hard upper limit on the size of the underlying DynamicData buffer.

If set to -1 (default): The buffer will grow unbounded to the size required to fit all members.

If set to any value other than -1: The buffer will not grow beyond this size. If setting a member's values requires the buffer to grow beyond this maximum, the member will fail to be set. If the buffer is required to grow beyond this maximum during deserialization, the sample will fail to be deserialized. The buffer_max_size cannot be smaller than the buffer_initial_size.

See also

rti::core::xtypes::DynamicDataProperty::buffer_initial_size (p. 1498)

8.102 rti::core::xtypes::DynamicDataTypeSerializationProperty Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Configures aspects of the memory management in the serialization of **dds::core::xtypes::DynamicData** (p. 1461) samples.

```
#include <rti/core/xtypes/DynamicDataProperty.hpp>
```

Public Member Functions

- **DynamicDataTypeSerializationProperty** ()

  Default configuration.
- **DynamicDataTypeSerializationProperty** (int32_t the_max_size_serialized, int32_t the_min_size_serialized, bool the_trim_to_size, bool the_skip_deserialization=false)

  Specifies all the serialization parameters.
- uint32_t **max_size_serialized** () const

  Getter (see setter with the same name)
- **DynamicDataTypeSerializationProperty** & **max_size_serialized** (uint32_t value)
- uint32_t **min_size_serialized** () const

  Getter (see setter with the same name)
- **DynamicDataTypeSerializationProperty** & **min_size_serialized** (uint32_t value)
- bool **trim_to_size** () const

  Getter (see setter with the same name)
- **DynamicDataTypeSerializationProperty** & **trim_to_size** (bool value)
- bool **skip_deserialization** () const

  Getter (see setter with the same name)
- **DynamicDataTypeSerializationProperty** & **skip_deserialization** (bool value)

Static Public Attributes

- static OMG_DDS_API_CLASS_VARIABLE const **DynamicDataTypeSerializationProperty DEFAULT**

  Default configuration.

8.102.1 Detailed Description

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Configures aspects of the memory management in the serialization of **dds::core::xtypes::DynamicData** (p. 1461) samples.

See also

rti::domain::register_type(dds::domain::DomainParticipant&,const std::string&, const dds::core↩ ::xtypes::DynamicType&, const rti::core::xtypes::DynamicDataTypeSerializationProperty&) (p. 733)

8.102.2 Constructor & Destructor Documentation

8.102.2.1 DynamicDataTypeSerializationProperty() [1/2]

```
rti::core::xtypes::DynamicDataTypeSerializationProperty::DynamicDataTypeSerializationProperty ( )
[inline]
```

Default configuration.

8.102.2.2 DynamicDataTypeSerializationProperty() [2/2]

```
rti::core::xtypes::DynamicDataTypeSerializationProperty::DynamicDataTypeSerializationProperty (
            int32_t the_max_size_serialized,
            int32_t the_min_size_serialized,
            bool the_trim_to_size,
            bool the_skip_deserialization = false )  [inline]
```

Specifies all the serialization parameters.

8.102.3 Member Function Documentation

8.102.3.1 max_size_serialized() [1/2]

```
uint32_t rti::core::xtypes::DynamicDataTypeSerializationProperty::max_size_serialized ( ) const
[inline]
```

Getter (see setter with the same name)

8.102.3.2 max_size_serialized() [2/2]

```
 DynamicDataTypeSerializationProperty & rti::core::xtypes::DynamicDataTypeSerializationProperty↩
::max_size_serialized (
            uint32_t value )  [inline]
```

This value is used to set the sizes of certain internal middleware buffers.

The effective value of the maximum serialized size will be the value of this field or the size automatically inferred from the type's **dds::core::xtypes::DynamicType** (p. 1504), whichever is smaller.

8.102.3.3 min_size_serialized() [1/2]

```
uint32_t rti::core::xtypes::DynamicDataTypeSerializationProperty::min_size_serialized ( ) const
[inline]
```

Getter (see setter with the same name)

8.102.3.4 min_size_serialized() [2/2]

```
DynamicDataTypeSerializationProperty & rti::core::xtypes::DynamicDataTypeSerializationProperty↩
::min_size_serialized (
            uint32_t value )  [inline]
```

This value is used to set the sizes of certain internal middleware buffers.

Default: 0xFFFFFFFF, a sentinel that indicates that this value must be equal to the value specified in max_size_↩ serialized.

8.102.3.5 trim_to_size() [1/2]

```
bool rti::core::xtypes::DynamicDataTypeSerializationProperty::trim_to_size ( ) const  [inline]
```

Getter (see setter with the same name)

8.102.3.6 trim_to_size() [2/2]

```
DynamicDataTypeSerializationProperty & rti::core::xtypes::DynamicDataTypeSerializationProperty↩
::trim_to_size (
            bool value )  [inline]
```

This property only applies to DynamicData samples that are obtained from the sample pool that is created by each DynamicData DataReader.

If set to 0 (default): The buffer will not be reallocated unless the deserialized size of the incoming sample is greater than the current buffer size.

If set to 1: The buffer of a DynamicData object obtained from the DDS sample pool will be freed and re-allocated for each sample to just fit the size of the deserialized data of the incoming sample. The newly allocated size will not be smaller than **rti::core::xtypes::DynamicDataProperty::buffer_initial_size** (p. 1498).

8.102.3.7 skip_deserialization() [1/2]

```
bool rti::core::xtypes::DynamicDataTypeSerializationProperty::skip_deserialization ( ) const  [inline]
```

Getter (see setter with the same name)

8.102.3.8 skip_deserialization() [2/2]

```
 DynamicDataTypeSerializationProperty & rti::core::xtypes::DynamicDataTypeSerializationProperty↩
::skip_deserialization (
            bool value )  [inline]
```

Setting this member to true boosts the performance for use cases in which the application does not require accessing the value of the different members of the received DynamicData object. Those use cases include recording, replaying, and bridging. You can access the CDR buffer of a DynamicData object by calling the method **dds::core::xtypes::↩ DynamicData::get_cdr_buffer** (p. 1481). This configuration option is only available in C, Traditional C++, Modern C++, and Python language bindings.

When this option is set to true, many of the DynamicData APIs cannot be used to access or manipulate the received DynamicData object. See each functions's documentation for whether or not the function is supported for DynamicData objects in CDR format.

See also

dds::core::xtypes::DynamicData::is_cdr (p. 1481)

dds::core::xtypes::DynamicData::get_cdr_buffer (p. 1481)

dds::core::xtypes::DynamicData::set_cdr_buffer (p. 1482)

[default] false

8.102.4 Member Data Documentation

8.102.4.1 DEFAULT

```
OMG_DDS_API_CLASS_VARIABLE const  DynamicDataTypeSerializationProperty rti::core::xtypes::Dynamic↩
DataTypeSerializationProperty::DEFAULT  [static]
```

Default configuration.

8.103 dds::core::xtypes::DynamicType Class Reference

<<***value-type***>> *(*p. *350)* Represents a runtime type.

`#include "dds/core/xtypes/DynamicType.hpp"`

Inheritance diagram for dds::core::xtypes::DynamicType:



Public Member Functions

- **dds::core::xtypes::TypeKind kind** () const
  Gets the type kind.
- std::string **name** () const
  Gets the name.
- bool **is_assignable_from** (const **DynamicType** &from) const
  Checks if this type is assignable from another type.

Related Functions

(Note that these are not member functions.)

- bool **is_primitive_type** (const **DynamicType** &t)
  *Determines if a **DynamicType** (*p. *1504) is a **PrimitiveType** (*p. *1959).*
- bool **is_constructed_type** (const **DynamicType** &t)
  *Determines if a **DynamicType** (*p. *1504) is a constructed type.*
- bool **is_collection_type** (const **DynamicType** &t)
  *Determines if a **DynamicType** (*p. *1504) is a **CollectionType** (*p. *939).*
- bool **is_aggregation_type** (const **DynamicType** &t)
  *Determines if a **DynamicType** (*p. *1504) is an aggregation type.*
- std::ostream & **operator<<** (std::ostream &out, const **DynamicType** &type)
  <<***extension***>> (p. *354) Converts the **DynamicType** (*p. *1504) to a string.*
- void **print_idl** (const **DynamicType** &type, unsigned int indent=0)
  <<***extension***>> (p. *354) Prints the IDL representation of this **DynamicType** (*p. *1504) to the standard output*
- std::ostream & **to_string** (std::ostream &out, const **DynamicType** &type, const **rti::core::xtypes::DynamicTypePrintFormatProperty** &format=DynamicTypePrintFormatProperty())
  <<***extension***>> (p. *354) Writes the string representation of this **DynamicType** (*p. *1504) to an output stream.*
- std::string **to_string** (const **DynamicType** &type, const **rti::core::xtypes::DynamicTypePrintFormatProperty** &format=DynamicTypePrintFormatProperty())
  <<***extension***>> (p. *354) Creates the string representation of this **DynamicType** (*p. *1504) to a string.*

8.103.1 Detailed Description

<<***value-type***>> *(p. 350)* Represents a runtime type.

A **DynamicType** (p. 1504) is a mechanism for representing a type at runtime. DynamicTypes allow the reflective manipulation of data (**DynamicData** (p. 1461)) and the inspection of the type information about discovered readers and writers.

You can obtain a **DynamicType** (p. 1504) in a few different ways:

1. Creating one by instantiating a subclass of **dds::core::xtypes::DynamicType** (p. 1504) instance, such as **dds::core::xtypes::StructType** (p. 2400) and adding members to it. For example:
```
using namespace dds::core::xtypes;
StructType point_type("Point");
point_type.add_member(Member("x", PrimitiveType<double>()));
point_type.add_member(Member("y", PrimitiveType<double>()));
```
2. From an IDL-generated type, using the **rti::topic::dynamic_type** (p. 1461) trait type. <<***extension***>> *(p. 354)*
3. From an XML description, using **dds::core::QosProvider::type()** (p. 2044).
4. Constructing it using tuples. <<***C++11***>> *(p. 353)* <<***experimental***>> *(p. 355)* <<***extension***>> *(p. 354)*
5. From a discovered DataReader or DataWriter, using the **built-in topics** (p. 46).

For examples, see **DynamicType and DynamicData Use Cases** (p. 609)

This an abstract base class of all the types. This class contains all of the internal state—derived classes don't add any new state. That means that it is safe to copy a **DynamicType** (p. 1504) without slicing.

To downcast a **DynamicType** (p. 1504) into a concrete class, you can check its **kind()** (p. 1506) and then apply a static_cast to the appropriate class.

For example, let's assume we have the following IDL type:
```
struct Bar {
    long x;
};
struct Foo {
    Bar bar;
    long y;
};
```

We obtain the type of **Foo** (p. 1588) and want to know the type of its member `bar`:
```
// Get the DynamicType of the IDL-generated type Foo
const StructType& foo_type = rti::topic::dynamic_type<Foo>::get();
// Get the type of its member bar
const DynamicType& bar_type = foo_type.member("bar").type();
// Check type kind and downcast
if (bar_type.kind() == TypeKind::STRUCTURE_TYPE) {
    const StructType& bar_struct = static_cast<const StructType&>(bar_type);
}
```

See also

dds::core::xtypes::DynamicData (p. 1461), the class that represents data samples for a **DynamicType** (p. 1504)

dds::topic::SubscriptionBuiltinTopicData::type() (p. 2434) and **dds::topic::PublicationBuiltinTopicData::type()** (p. 1985) to obtain the **type** (p. 2044) of a remote subscription or publication

rti::topic::dynamic_type (p. 1461) to obtain the equivalent **DynamicType** (p. 1504) of an IDL-generated **type** (p. 2044)

DynamicType and DynamicData Use Cases (p. 609)

8.103.2 Member Function Documentation

8.103.2.1 kind()

```
dds::core::xtypes::TypeKind dds::core::xtypes::DynamicType::kind ( ) const
```

Gets the type kind.

8.103.2.2 name()

```
std::string dds::core::xtypes::DynamicType::name ( ) const
```

Gets the name.

8.103.2.3 is_assignable_from()

```
bool dds::core::xtypes::DynamicType::is_assignable_from (
            const  DynamicType & from ) const
```

Checks if this type is assignable from another type.

If this type is assignable from the `from` type, it means that it is possible, in a structural way, to set the contents of an object of this type to the contents of an object of type `from` (or a subset of those contents) without leading to incorrect interpretations of that information.

For additional information on type assignability, refer to the `OMG Extensible and Dynamic Topic Types for DDS Specification`.

MT Safety:

SAFE.

Parameters

from	<<***in***>> *(p. 355)* The type to check assignability from.

Returns

true if this type is assignable from `from`. Otherwise, false.

8.103.3 Friends And Related Function Documentation

8.103.3.1 is_primitive_type()

```
bool is_primitive_type (
            const  DynamicType & t )  [related]
```

Determines if a **DynamicType** (p. 1504) is a **PrimitiveType** (p. 1959).

8.103.3.2 is_constructed_type()

```
bool is_constructed_type (
            const  DynamicType & t )  [related]
```

Determines if a **DynamicType** (p. 1504) is a constructed type.

This includes **EnumType** (p. 1535), **AliasType** (p. 801), collection types and aggregation types.

See also

is_collection_type (p. 1507)

is_aggregation_type (p. 1507)

8.103.3.3 is_collection_type()

```
bool is_collection_type (
            const  DynamicType & t )  [related]
```

Determines if a **DynamicType** (p. 1504) is a **CollectionType** (p. 939).

This includes **ArrayType** (p. 828), **SequenceType** (p. 2344), **StringType** (p. 2399), and **WStringType** (p. 2665).

8.103.3.4 is_aggregation_type()

```
bool is_aggregation_type (
            const  DynamicType & t )  [related]
```

Determines if a **DynamicType** (p. 1504) is an aggregation type.

This includes **StructType** (p. 2400) and **UnionType** (p. 2584).

8.103.3.5 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const  DynamicType & type )  [related]
```

<<***extension***>> *(p. 354)* Converts the **DynamicType** (p. 1504) to a string.

This operator writes the **DynamicType** (p. 1504) type to the ostream out using default values for **DynamicTypePrint↩ FormatProperty** (p. 1509) and returns a reference to out.

8.103.3.6 print_idl()

```
void print_idl (
            const  DynamicType & type,
            unsigned int indent = 0 )  [related]
```

<<***extension***>> *(p. 354)* Prints the IDL representation of this **DynamicType** (p. 1504) to the standard output

8.103.3.7 to_string() [1/2]

```
std::ostream & to_string (
            std::ostream & out,
            const  DynamicType & type,
            const  rti::core::xtypes::DynamicTypePrintFormatProperty & format = DynamicTypePrintFormatProperty()
)  [related]
```

<<***extension***>> *(p. 354)* Writes the string representation of this **DynamicType** (p. 1504) to an output stream.

Parameters

out	The output stream to which the string representation of the **DynamicType** (p. 1504) should be printed.
type	The **DynamicType** (p. 1504) to be printed.
format	The **DynamicTypePrintFormatProperty** (p. 1509) used to define the format of the string.

Returns

A reference to out.

8.103.3.8 to_string() [2/2]

```
std::string to_string (
            const  DynamicType & type,
```

```
            const rti::core::xtypes::DynamicTypePrintFormatProperty & format = DynamicTypePrintFormatProperty()
) [related]
```

<<***extension***>> *(p. 354)* Creates the string representation of this **DynamicType** (p. 1504) to a string.

Parameters

type	The **DynamicType** (p. 1504) to be printed.
format	The **DynamicTypePrintFormatProperty** (p. 1509) used to define the format of the string.

Returns

The string representation of the **DynamicType** (p. 1504).

8.104 rti::core::xtypes::DynamicTypePrintFormatProperty Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* A collection of attributes used to configure how DynamicTypes will be formatted when converted to strings.

```
#include <rti/core/xtypes/DynamicTypePrintFormat.hpp>
```

Public Member Functions

- **DynamicTypePrintFormatProperty** (unsigned int indent_in=0, bool print_ordinals_in=false, const **DynamicTypePrintKind** print_kind_in= **rti::core::xtypes::DynamicTypePrintKind::idl**, bool print_complete_type_in=false)

  Initializes the properties.
- unsigned int **indent** () const

  Get the value of indent.
- **DynamicTypePrintFormatProperty** & **indent** (unsigned int value)

  Set the amount of additional indent to be included when converting a DynamicType to a string.
- bool **print_ordinals** () const

  Get the value of print_ordinals.
- **DynamicTypePrintFormatProperty** & **print_ordinals** (bool value)

  Set whether or not to print ordinal values when printing EnumTypes.
- **DynamicTypePrintKind** **print_kind** () const

  Get the value of print_kind.
- **DynamicTypePrintFormatProperty** & **print_kind** (**DynamicTypePrintKind** value)

  Sets the format to use when printing the TypeCode.
- bool **print_complete_type** () const

  Get the value of print_complete_type.
- **DynamicTypePrintFormatProperty** & **print_complete_type** (bool value)

  Set whether or not to print the complete type.

8.104.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* A collection of attributes used to configure how DynamicTypes will be formatted when converted to strings.

See also

rti::core::xtypes::to_string

8.104.2 Constructor & Destructor Documentation

8.104.2.1 DynamicTypePrintFormatProperty()

```
rti::core::xtypes::DynamicTypePrintFormatProperty::DynamicTypePrintFormatProperty (
            unsigned int indent_in = 0,
            bool print_ordinals_in = false,
            const  DynamicTypePrintKind print_kind_in =  rti::core::xtypes::DynamicTypePrint↩
Kind::idl,
            bool print_complete_type_in = false )  [inline], [explicit]
```

Initializes the properties.

8.104.3 Member Function Documentation

8.104.3.1 indent() [1/2]

```
unsigned int rti::core::xtypes::DynamicTypePrintFormatProperty::indent ( ) const  [inline]
```

Get the value of indent.

See also

indent(unsigned int value) (p. 1510)

8.104.3.2 indent() [2/2]

```
DynamicTypePrintFormatProperty & rti::core::xtypes::DynamicTypePrintFormatProperty::indent (
            unsigned int value )  [inline]
```

Set the amount of additional indent to be included when converting a DynamicType to a string.

Configures how much additional indent is applied when converting a TypeCode to a string. This value acts as a total offset on the string, increasing the indent applied to all elements by the same amount. With an indent of 0, a string representation of a TypeCode may appear as:

```
struct myType {
    long x;
};
```

Using an indent of 1, the same TypeCode would be printed as:

```
    struct myType {
        long x;
    };
```

I.e., the entire structure is indented.

Parameters

value	The value to set for indent

8.104.3.3 print_ordinals() [1/2]

```
bool rti::core::xtypes::DynamicTypePrintFormatProperty::print_ordinals ( ) const  [inline]
```

Get the value of print_ordinals.

See also

print_ordinals(bool value) (p. 1511)

8.104.3.4 print_ordinals() [2/2]

```
DynamicTypePrintFormatProperty & rti::core::xtypes::DynamicTypePrintFormatProperty::print_↩
ordinals (
            bool value )  [inline]
```

Set whether or not to print ordinal values when printing EnumTypes.

When set to true, the ordinal value of each enumerator within an enum will be printed, otherwise only non-default ordinals are printed. Take for example the following enum:

```
enum myEnum {
    RED,
    GREEN = 3,
    BLUE,
};
```

When print_ordinals is set to false it would be printed as:

```
enum myEnum {
    RED,
    GREEN = 3,
    BLUE,
};
```

But with print_ordinals set to true it would be printed as:

```
enum myEnum {
    RED = 0,
    GREEN = 3,
    BLUE = 4,
};
```

Parameters

value	The value to set for print_ordinals

8.104.3.5 print_kind() [1/2]

```
DynamicTypePrintKind rti::core::xtypes::DynamicTypePrintFormatProperty::print_kind ( ) const
```

[inline]

Get the value of print_kind.

See also

print_kind(DynamicTypePrintKind value) (p. 1513)

8.104.3.6 print_kind() [2/2]

DynamicTypePrintFormatProperty & rti::core::xtypes::DynamicTypePrintFormatProperty::print_kind (**DynamicTypePrintKind** *value*) [inline]

Sets the format to use when printing the TypeCode.

When print_kind is DDS_TYPE_CODE_PRINT_KIND_IDL, the type will be printed in IDL format. For example:

```
struct Foo {
    float32 bar;
};
```

When print_kind is DDS_TYPE_CODE_PRINT_KIND_XML, the type will be printed in XML format. For example:

```
<struct name="Foo">
    <member name="bar" type="float32"/>
</struct>
```

Parameters

value	The value to set for print_kind

8.104.3.7 print_complete_type() [1/2]

bool rti::core::xtypes::DynamicTypePrintFormatProperty::print_complete_type ( ) const [inline]

Get the value of print_complete_type.

See also

print_complete_type(bool value) (p. 1513)

8.104.3.8 print_complete_type() [2/2]

```
DynamicTypePrintFormatProperty & rti::core::xtypes::DynamicTypePrintFormatProperty::print_↩
complete_type (
            bool value )  [inline]
```

Set whether or not to print the complete type.

When print_complete_type is true, the complete type will be printed. When print_complete_type is false, only the top level will be printed.

Take for example the following types:

```
struct Foo {
    float32 member;
};
struct Bar {
    Foo foo;
};
```

When print_complete_type is false, this is printed as:

```
struct Bar {
    Foo foo;
};
```

When print_complete_type is true, this is printed as:

```
struct Foo {
    float32 member;
};
struct Bar {
    Foo foo;
};
```

Parameters

value	The value to set for print_complete_type

8.105 rti::core::EndpointGroup Class Reference

<<***extension***>> *(p. 354)* Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **EndpointGroup** (const std::string &the_role_name, int32_t the_quorum_count)

  Creates an instance with a role name and a quorum.
- std::string **role_name** () const

  Getter (see setter with the same name)
- **EndpointGroup** & **role_name** (const std::string &the_role_name)

  Defines the role name of the endpoint group.
- int32_t **quorum_count** () const

  Getter (see setter with the same name)
- **EndpointGroup** & **quorum_count** (int32_t the_quorum_count)

  Defines the minimum number of members that satisfies the endpoint group.

8.105.1 Detailed Description

<<***extension***>> *(p. 354)* Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.

8.105.2 Constructor & Destructor Documentation

8.105.2.1 EndpointGroup()

```
rti::core::EndpointGroup::EndpointGroup (
            const std::string & the_role_name,
            int32_t the_quorum_count )  [inline]
```

Creates an instance with a role name and a quorum.

8.105.3 Member Function Documentation

8.105.3.1 role_name() [1/2]

```
std::string rti::core::EndpointGroup::role_name ( ) const  [inline]
```

Getter (see setter with the same name)

8.105.3.2 role_name() [2/2]

```
EndpointGroup & rti::core::EndpointGroup::role_name (
            const std::string & the_role_name )  [inline]
```

Defines the role name of the endpoint group.

If used in the **rti::core::policy::Availability** (p. 868) on a **dds::pub::DataWriter** (p. 1138), it specifies the name that identifies a Durable Subcription.

The role name can be at most 255 characters in length.

8.105.3.3 quorum_count() [1/2]

```
int32_t rti::core::EndpointGroup::quorum_count ( ) const  [inline]
```

Getter (see setter with the same name)

8.105.3.4 quorum_count() [2/2]

```
EndpointGroup & rti::core::EndpointGroup::quorum_count (
            int32_t the_quorum_count )  [inline]
```

Defines the minimum number of members that satisfies the endpoint group.

If used in the **rti::core::policy::Availability** (p. 868) on a **dds::pub::DataWriter** (p. 1138), it specifies the number of DataReaders that must acknowledge a sample before the sample is considered to be acknowledged by the Durable Subscription.

8.106 rti::topic::trust::EndpointTrustAlgorithmInfo Class Reference

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins algorithm information associated with the discovered endpoint.

```
#include <rti/topic/trust/EndpointTrustAlgorithmInfo.hpp>
```

Public Member Functions

- **EndpointTrustAlgorithmInfo** ()=default

  Create an instance with the default Trust Algorithm Info associated with the discovered endpoint.
- **EndpointTrustInterceptorAlgorithmInfo interceptor** () const

  Get the Trust Plugins algorithms for interception of data and metadata exchanged by the endpoint.

8.106.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins algorithm information associated with the discovered endpoint.

8.106.2 Constructor & Destructor Documentation

8.106.2.1 EndpointTrustAlgorithmInfo()

```
rti::topic::trust::EndpointTrustAlgorithmInfo::EndpointTrustAlgorithmInfo ( )  [default]
```

Create an instance with the default Trust Algorithm Info associated with the discovered endpoint.

The meaning of this field may vary depending on what Trust Plugins the endpoint is using.

8.106.3 Member Function Documentation

8.106.3.1 interceptor()

```
EndpointTrustInterceptorAlgorithmInfo rti::topic::trust::EndpointTrustAlgorithmInfo::interceptor
( ) const  [inline]
```

Get the Trust Plugins algorithms for interception of data and metadata exchanged by the endpoint.

8.107 rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo Class Reference

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins interception algorithm information associated with the discovered endpoint.

```
#include <rti/topic/trust/EndpointTrustInterceptorAlgorithmInfo.hpp>
```

Public Member Functions

- **EndpointTrustInterceptorAlgorithmInfo** ()=default

  Create an instance with the default Trust Interceptor Algorithm Info associated with the discovered endpoint.
- uint32_t **required_mask** () const

  Get the Trust Plugins algorithms used for interception of data and metadata exchanged by the endpoint.
- uint32_t **supported_mask** () const

  Get the Trust Plugins algorithms supported for interception of data and metadata exchanged by the endpoint.

8.107.1 Detailed Description

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Trust Plugins interception algorithm information associated with the discovered endpoint.

8.107.2 Constructor & Destructor Documentation

8.107.2.1 EndpointTrustInterceptorAlgorithmInfo()

```
rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo::EndpointTrustInterceptorAlgorithmInfo (
) [default]
```

Create an instance with the default Trust Interceptor Algorithm Info associated with the discovered endpoint.

The meaning of this field may vary depending on what Trust Plugins the endpoint is using.

8.107.3 Member Function Documentation

8.107.3.1 required_mask()

```
uint32_t rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo::required_mask ( ) const [inline]
```

Get the Trust Plugins algorithms used for interception of data and metadata exchanged by the endpoint.

8.107.3.2 supported_mask()

```
uint32_t rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo::supported_mask ( ) const [inline]
```

Get the Trust Plugins algorithms supported for interception of data and metadata exchanged by the endpoint.

8.108 rti::topic::trust::EndpointTrustProtectionInfo Class Reference

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins Protection information associated with the discovered endpoint.

`#include <rti/topic/trust/EndpointTrustProtectionInfo.hpp>`

Public Member Functions

- **EndpointTrustProtectionInfo** ()=default

  Create an instance with the default Trust Plugins Protection Info associated with the discovered endpoint.
- uint32_t **bitmask** () const

  Information about how RTPS wire serialization, discovery, and liveliness interact with the associated Trust Plugins Protection configuration.
- uint32_t **plugin_bitmask** () const

  Internal plugin information that is opaque to DDS.

8.108.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins Protection information associated with the discovered endpoint.

8.108.2 Constructor & Destructor Documentation

8.108.2.1 EndpointTrustProtectionInfo()

`rti::topic::trust::EndpointTrustProtectionInfo::EndpointTrustProtectionInfo ( ) [default]`

Create an instance with the default Trust Plugins Protection Info associated with the discovered endpoint.

The meaning of this field may vary depending on what Trust Plugins the endpoint is using.

8.108.3 Member Function Documentation

8.108.3.1 bitmask()

```
uint32_t rti::topic::trust::EndpointTrustProtectionInfo::bitmask ( ) const  [inline]
```

Information about how RTPS wire serialization, discovery, and liveliness interact with the associated Trust Plugins Protection configuration.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.108.3.2 plugin_bitmask()

```
uint32_t rti::topic::trust::EndpointTrustProtectionInfo::plugin_bitmask ( ) const  [inline]
```

Internal plugin information that is opaque to DDS.

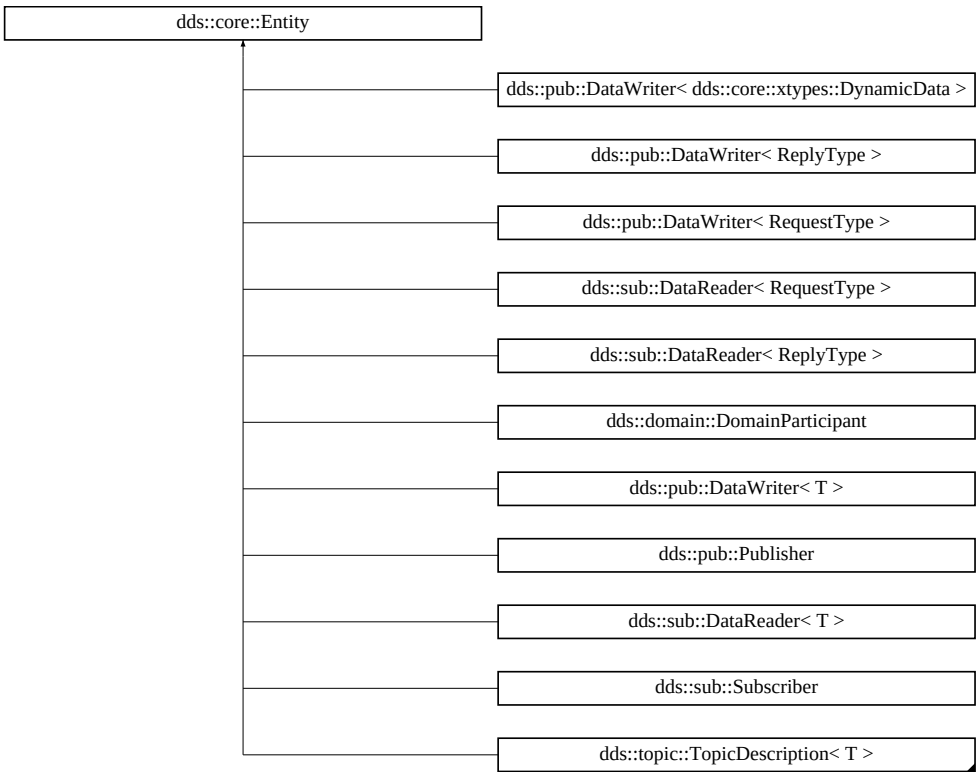
The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.109 dds::core::Entity Class Reference

<<***reference-type***>> *(p. 351)* This is the abstract base class for all the DDS objects that support QoS policies, a listener and a status condition.

`#include <dds/core/Entity.hpp>`

Inheritance diagram for dds::core::Entity:



Public Member Functions

- void **enable** ()

  Enables this entity (if it was created disabled)
- const **dds::core::status::StatusMask status_changes** ()

  Retrieves the list of communication statuses that are triggered.
- const **dds::core::InstanceHandle instance_handle** () const

  Get the instance handle that represents this entity.
- void **close** ()

  Forces the destruction of this entity.
- void **retain** ()

  Disables the automatic destruction of this entity.

Related Functions

(Note that these are not member functions.)

- template<typename TO , typename FROM >
  TO **polymorphic_cast** (const FROM &from)

  *Downcasts an **Entity** (p. 1520) to a subclass.*

8.109.1 Detailed Description

<<***reference-type***>> *(p. 351)* This is the abstract base class for all the DDS objects that support QoS policies, a listener and a status condition.

Note

Entity (p. 1520) and its subclasses provides all the functions of a <<***reference-type***>> *(p. 351)*, including **close()** (p. 1525) and **retain()** (p. 1526).

See also

Entity Use Cases (p. 325)

All operations except for set_qos(), get_qos(), set_listener(), get_listener() and **enable()** (p. 1524), may return the value **dds::core::NotEnabledError** (p. 1869).

QoS:

All QoS Policies (p. 508)

Status:

Status Kinds (p. 429)

Listener:

Listener (p. 1637)

8.109.2 Abstract operations

Each derived entity provides the following operations specific to its role in RTI Connext. Note that these are not member functions of **dds::core::Entity** (p. 1520), but members of each of the derived classes.

8.109.2.1 set_qos (abstract)

This operation sets the QoS policies of the **dds::core::Entity** (p. 1520). For example, see **dds::domain::DomainParticipant::qos(const dds::domain::qos::DomainParticipantQos&)** (p. 1337).

Precondition

Certain policies are immutable (see **All QoS Policies** (p. 508)): they can only be set at **dds::core::Entity** (p. 1520) creation time or before the entity is enabled. If `set_qos()` is invoked after the **dds::core::Entity** (p. 1520) is enabled and it attempts to change the value of an immutable policy, the operation will fail and return **dds::core::ImmutablePolicyError** (p. 1609).

Certain values of QoS policies can be incompatible with the settings of the other policies. The `set_qos()` operation will also fail if it specifies a set of values that, once combined with the existing values, would result in an inconsistent set of policies. In this case, the operation will fail and return **dds::core::InconsistentPolicyError** (p. 1610).

If the application supplies a non-default value for a QoS policy that is not supported by the implementation of the service, the set_qos operation will fail and return **dds::core::UnsupportedError** (p. 2590).

Postcondition

The existing set of policies is only changed if the `set_qos()` operation succeeds.

Possible exceptions thrown in addition to **Standard Exceptions** (p. 428) : **dds::core::ImmutablePolicyError** (p. 1609), **dds::core::InconsistentPolicyError** (p. 1610), **dds::core::NotAllowedBySecurityError** (p. 1868).

8.109.2.1.1 Default QoS When you create an **Entity** (p. 1520) the Qos parameter is optional. When you don't specify it the *default QoS* applies. Applications can configure this default value.

If the application does nothing explicit to specify the *default QoS*, the default RTI Connext values will be used whenever an entity is created. Each of the QoS policies and their defaults can be found here: **All QoS Policies** (p. 508).

```
// Create a participant with default QoS

dds::domain::DomainParticipant dp_default_qos(0);

// Create a participant with specific QoS

dds::domain::qos::DomainParticipantQos participant_qos;

// Set some values in participant_qos

...

dds::domain::DomainParticipant dp_custom_qos(0, participant_qos);
```

For each entity, there are `default_*_qos` member functions that allow an application to set the default QoS values for all entities that are created from that entity. For example, **dds::domain::DomainParticipant::default_subscriber_qos** (p. 1343) sets the default **dds::sub::qos::SubscriberQos** (p. 2423) that will be used for each **dds::sub::Subscriber** (p. 2410) that is created from that DomainParticipant.

```
dds::domain::DomainParticipant participant(0);
// Set the default qos for any subscriber that is created from this participant
dds::sub::qos::SubscriberQos subscriber_qos;
// Set some values in subscriber_qos
subscriber_qos.policy<dds::core::policy::Partition>(...);
// or get it from a QosProvider:
subscriber_qos = dds::core::QosProvider::Default().subscriber_qos(...);
participant.default_subscriber_qos(subscriber_qos);
```

See also

dds::core::QosProvider (p. 2025)

8.109.2.2 get_qos (abstract)

This operation allows access to the existing set of QoS policies for the **dds::core::Entity** (p. 1520). For example, see **dds::domain::DomainParticipant::qos() const** (p. 1336)

8.109.2.3 set_listener (abstract)

This operation installs a **Listener** (p. 1637) on the **dds::core::Entity** (p. 1520). The listener will only be invoked on the changes of communication status indicated by the specified `mask`.

There are two components involved when setting up listeners: the listener itself and the mask. Both of these can be NULL.

Listeners for some Entities derive from the Connext DDS Listeners for related Entities. This means that the derived **Listener** (p. 1637) has all of the methods of its parent class. You can install Listeners at all levels of the object hierarchy. At the top is the DomainParticipantListener; only one can be installed in a DomainParticipant. Then every Subscriber and Publisher can have their own **Listener** (p. 1637). Finally, each Topic, DataReader and DataWriter can have their own listeners. All are optional.

Suppose, however, that an **Entity** (p. 1520) does not install a **Listener** (p. 1637), or installs a **Listener** (p. 1637) that does not have particular communication status selected in the bitmask. In this case, if/when that particular status changes for that **Entity** (p. 1520), the corresponding **Listener** (p. 1637) for that **Entity** (p. 1520)'s parent is called. Status changes are "propagated" from child **Entity** (p. 1520) to parent **Entity** (p. 1520) until a **Listener** (p. 1637) is found that is registered for that status. Connext DDS will give up and drop the status-change event only if no Listeners have been installed in the object hierarchy to be called back for the specific status.

The following table describes the effect of different combinations of Listeners and Status Bit Masks considering the hierarchical processing.

Table 8.512 Effect of Different Combinations of Listeners and Status Bit Masks

	No Bits Set in Mask	Some/All Bits Set in Mask
Listener (p. 1637) is Specified	Connext DDS finds the next most relevant listener for the changed status	For the statuses that are enabled in the mask, the most relevant listener will be called. The 'statusChangedFlag' for the relevant status is reset
Listener (p. 1637) is NULL	Connext DDS behaves as if the listener is not installed and finds the next most relevant listener for that status	Connext DDS behaves as if the listener is installed, but the callback is doing nothing. This is called a "nil" listener

Postcondition

Only one listener can be attached to each **dds::core::Entity** (p. 1520). If a listener was already set, the operation `set_listener()` will replace it with the new one. Consequently, if the value NULL is passed for the listener parameter to the set_listener operation, any existing listener will be removed.

8.109.2.4 get_listener (abstract)

This operation allows access to the existing **Listener** (p. 1637) attached to the **dds::core::Entity** (p. 1520).

If no listener is installed on the **dds::core::Entity** (p. 1520), this operation will return NULL.

8.109.3 Member Function Documentation

8.109.3.1 enable()

```
void dds::core::Entity::enable ( )  [inline]
```

Enables this entity (if it was created disabled)

This operation enables the **Entity** (p. 1520). **Entity** (p. 1520) objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 532) QoS policy on the corresponding factory for the **dds::core::Entity** (p. 1520).

By default, **ENTITY_FACTORY** (p. 532) is set so that it is not necessary to explicitly call **dds::core::Entity::enable** (p. 1524) on newly created entities.

The **dds::core::Entity::enable** (p. 1524) operation is idempotent. Calling enable on an already enabled **Entity** (p. 1520) returns OK and has no effect.

If a **dds::core::Entity** (p. 1520) has not yet been enabled, the following kinds of operations may be invoked on it:

- set or get the QoS policies (including default QoS policies) and listener
- **dds::core::cond::StatusCondition::StatusCondition(const dds::core::Entity&)** (p. 2373)
- **Entity** (p. 1520) constructors
- **dds::core::Entity::status_changes()** (p. 1525) and other get status operations (although the status of a disabled entity never changes)
- find operations

Other operations may explicitly state that they may be called on disabled entities; those that do not will return the error **dds::core::NotEnabledError** (p. 1869).

It is legal to delete an **dds::core::Entity** (p. 1520) that has not been enabled .

Entities created from a factory **Entity** (p. 1520) that is disabled are created disabled, regardless of the setting of the **dds::core::policy::EntityFactory** (p. 1527).

Calling enable on an **Entity** (p. 1520) whose factory **Entity** (p. 1520) is not enabled will fail and return **dds::core::PreconditionNotMetError** (p. 1941).

If **dds::core::policy::EntityFactory::autoenable_created_entities** (p. 1529) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory (for example, enabling a **dds::pub::Publisher** (p. 1993) will enable all its contained **dds::pub::DataWriter** (p. 1138) objects)

Listeners associated with an entity are not called until the entity is enabled.

Conditions associated with a disabled entity are "inactive," that is, they have a `trigger_value` == FALSE.

Exceptions

One	of the **Standard Exceptions** (p. 428), **Standard Exceptions** (p. 428) or **dds::core::PreconditionNotMetError** (p. 1941).

8.109.3.2 status_changes()

```
const dds::core::status::StatusMask dds::core::Entity::status_changes ( ) [inline]
```

Retrieves the list of communication statuses that are triggered.

That is, the list of statuses whose value has changed since the last time the application read the status using the get_∗_status() method.

When the entity is first created or if the entity is not enabled, all communication statuses are in the "untriggered" state so the list returned by the get_status_changes operation will be empty.

The list of statuses returned by the get_status_changes operation refers to the status that are triggered on the **Entity** (p. 1520) itself and does not include statuses that apply to contained entities.

Returns

list of communication statuses in the **dds::core::Entity** (p. 1520) that are triggered.

See also

Status Kinds (p. 429)

8.109.3.3 instance_handle()

```
const dds::core::InstanceHandle dds::core::Entity::instance_handle ( ) const [inline]
```

Get the instance handle that represents this entity.

This operation returns the **dds::core::InstanceHandle** (p. 1612) that represents the **dds::core::Entity** (p. 1520).

Returns

the instance handle associated with this entity.

8.109.3.4 close()

```
void dds::core::Entity::close ( )  [inline]
```

Forces the destruction of this entity.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

See also

Reference types (p. 351)

8.109.3.5 retain()

```
void dds::core::Entity::retain ( )  [inline]
```

Disables the automatic destruction of this entity.

See also

Reference types (p. 351)

8.109.4 Friends And Related Function Documentation

8.109.4.1 polymorphic_cast()

```
template<typename TO , typename FROM >
TO polymorphic_cast (
             const FROM & from )  [related]
```

Downcasts an **Entity** (p. 1520) to a subclass.

Note

- Header: <dds/core/ref_traits.hpp>
- Namespace: **dds::core** (p. 616)

Template Parameters

TO	The type to cast to
FROM	The type to cast from

Parameters

from	The object to cast to type `TO`

Returns

A reference to the same object `from`, but cast to the type `TO`. Both objects share ownership of the underlying entity.

For example:

```
using namespace dds::core;
using namespace dds::domain;
DomainParticipant participant(0);
Entity entity = participant; // Assignment to base
DomainParticipant participant2 = polymorphic_cast<DomainParticipant>(entity);
assert (participant == participant2); // Same reference
```

See also

Reference types (p. 351)

8.110 dds::core::policy::EntityFactory Class Reference

Configures a **dds::core::Entity** (p. 1520) that acts as factory of other entities.

`#include <dds/core/policy/CorePolicy.hpp>`

Public Member Functions

- **TEntityFactory** ()

  Default policy.
- **TEntityFactory** (bool the_auto_enable)

  Specifies whether the entity acting as a factory automatically enables the instances it creates.
- **TEntityFactory** & **autoenable_created_entities** (bool on)

  Specifies whether the entity acting as a factory automatically enables the instances it creates.
- bool **autoenable_created_entities** () const

  Getter (see setter with the same name)

Static Public Member Functions

- static **TEntityFactory** **AutoEnable** ()

  Creates `EntityFactory(true)`
- static **TEntityFactory** **ManuallyEnable** ()

  Creates `EntityFactory(false)`

8.110.1 Detailed Description

Configures a **dds::core::Entity** (p. 1520) that acts as factory of other entities.

Entity:

DomainParticipantFactory, **dds::domain::DomainParticipant** (p. 1326), **dds::pub::Publisher** (p. 1993), **dds::sub::Subscriber** (p. 2410)

Properties:

RxO (p. 514) = NO
Changeable (p. 514) = **YES** (p. 514)

8.110.2 Usage

This policy controls the behavior of the **dds::core::Entity** (p. 1520) as a factory for other entities. It controls whether or not child entities are created in the enabled state.

RTI Connext uses a factory design pattern for creating DDS Entities. That is, a parent entity must be used to create child entities. DomainParticipants create Topics, Publishers and Subscribers. Publishers create DataWriters. Subscribers create DataReaders.

By default, a child object is enabled upon creation (initialized and may be actively used). With this QoS policy, a child object can be created in a disabled state. A disabled entity is only partially initialized and cannot be used until the entity is enabled. Note: an entity can only be *enabled*; it cannot be *disabled* after it has been enabled.

This QoS policy is useful to synchronize the initialization of DDS Entities. For example, when a **dds::sub::DataReader** (p. 976) is created in an enabled state, its existence is immediately propagated for discovery and the **dds::sub::DataReader** (p. 976) object's listener called as soon as data is received. The initialization process for an application may extend beyond the creation of the **dds::sub::DataReader** (p. 976), and thus, it may not be desireable for the **dds::sub::DataReader** (p. 976) to start to receive or process any data until the initialization process is complete. So by creating readers in a disabled state, your application can make sure that no data is received until the rest of the application initialization is complete, and at that time, enable the them.

Note: if an entity is disabled, then all of the child entities it creates will be disabled, too, regardless of the setting of this QoS policy. However, enabling a disabled entity will enable all of its children if this QoS policy is set to automatically enable children entities.

This policy is mutable. A change in the policy affects only the entities created after the change, not any previously created entities.

See also

UserData (p. 2591)

8.110.3 Member Function Documentation

8.110.3.1 TEntityFactory() [1/2]

```
dds::core::policy::EntityFactory::TEntityFactory ( )  [inline]
```

Default policy.

8.110.3.2 TEntityFactory() [2/2]

```
dds::core::policy::EntityFactory::TEntityFactory (
            bool the_auto_enable )  [inline], [explicit]
```

Specifies whether the entity acting as a factory automatically enables the instances it creates.

8.110.3.3 autoenable_created_entities() [1/2]

```
TEntityFactory & dds::core::policy::EntityFactory::autoenable_created_entities (
            bool on )  [inline]
```

Specifies whether the entity acting as a factory automatically enables the instances it creates.

8.110.3.4 autoenable_created_entities() [2/2]

```
bool dds::core::policy::EntityFactory::autoenable_created_entities ( ) const  [inline]
```

Getter (see setter with the same name)

8.110.3.5 AutoEnable()

```
static  TEntityFactory dds::core::policy::EntityFactory::AutoEnable ( )  [inline], [static]
```

Creates `EntityFactory(true)`

8.110.3.6 ManuallyEnable()

```
static  TEntityFactory dds::core::policy::EntityFactory::ManuallyEnable ( )  [inline], [static]
```

Creates `EntityFactory(false)`

8.111 rti::core::policy::EntityName Class Reference

<<***extension***>> *(p. 116)* Assigns a name to a DomainParticipant, Publisher, Subscriber, DataWriter or DataReader.

`#include <rti/core/policy/CorePolicy.hpp>`

Public Member Functions

- **EntityName** ()
  Creates the default policy (no name)
- **EntityName** (const std::string &the_name)
  Creates an instance that specifies an entity name.
- **EntityName** & **name** (const **rti::core::optional_value**< std::string > &the_name)
  Sets the entity name.
- **EntityName** & **name** (const char *the_name)
  Sets the entity name.
- **rti::core::optional_value**< std::string > **name** () const
  Gets the entity name.
- **EntityName** & **role_name** (const **rti::core::optional_value**< std::string > &the_name)
  Specifies an entity role name.
- **EntityName** & **role_name** (const char *the_name)
  Sets the role name.
- **rti::core::optional_value**< std::string > **role_name** () const
  Gets the entity role name.

8.111.1 Detailed Description

<<***extension***>> *(p. 116)* Assigns a name to a DomainParticipant, Publisher, Subscriber, DataWriter or DataReader.

Except for the Publisher and Subscriber, these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.

Entity:

dds::domain::DomainParticipant (p. 1326), **dds::sub::Subscriber** (p. 2410), **dds::pub::Publisher** (p. 1993), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Properties:

RxO (p. 514) = NO;
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

8.111.2 Usage

The name and role name can be at most 255 characters in length.

8.111.3 Constructor & Destructor Documentation

8.111.3.1 EntityName() [1/2]

```
rti::core::policy::EntityName::EntityName ( )  [inline]
```

Creates the default policy (no name)

8.111.3.2 EntityName() [2/2]

```
rti::core::policy::EntityName::EntityName (
            const std::string & the_name )  [inline], [explicit]
```

Creates an instance that specifies an entity name.

8.111.4 Member Function Documentation

8.111.4.1 name() [1/3]

```
EntityName & rti::core::policy::EntityName::name (
            const  rti::core::optional_value< std::string > & the_name )
```

Sets the entity name.

Parameters

the_name	An optional string. An unset value indicates that this entity should have no name. A set value will set a name. The string can't exceed 255 characters.

[default] Unset (no name)

8.111.4.2 name() [2/3]

```
EntityName & rti::core::policy::EntityName::name (
            const char * the_name )
```

Sets the entity name.

Parameters

the_name	Can't exceed 255 characters.

8.111.4.3 name() [3/3]

```
rti::core::optional_value< std::string > rti::core::policy::EntityName::name ( ) const
```

Gets the entity name.

Returns

An optional string. An unset value indicates that the entity has not been assigned a name.

8.111.4.4 role_name() [1/3]

```
EntityName & rti::core::policy::EntityName::role_name (
            const  rti::core::optional_value< std::string > & the_name )
```

Specifies an entity role name.

With Durable Subscriptions this name is used to specify to which Durable Subscription the DataReader belongs.

With Collaborative DataWriters this name is used to specify to which endpoint group the DataWriter belongs.

Parameters

the_name	An optional string. An unset value indicates that the Entity will have no role name (the default). The string can't exceed 255 characters.

[default] Unset (no role name)

8.111.4.5 role_name() [2/3]

```
EntityName & rti::core::policy::EntityName::role_name (
            const char * the_name )
```

Sets the role name.

See also

role_name(const rti::core::optional_value<std::string>&) (p. 1532)

8.111.4.6 role_name() [3/3]

```
rti::core::optional_value< std::string > rti::core::policy::EntityName::role_name ( ) const
```

Gets the entity role name.

Returns

An optional string. An unset value indicates that the entity has not been assigned a role name.

8.112 dds::core::xtypes::EnumMember Class Reference

<<***value-type***>> *(*p. *350)* Represents a **EnumType** (p. 1535) member

`#include "dds/core/xtypes/MemberType.hpp"`

Public Member Functions

- **EnumMember** (const std::string & **name**, int32_t **ordinal**)
- const **dds::core::string** & **name** () const

  Gets the name.
- **dds::core::string** & **name** ()

  Gets the member name.
- **EnumMember** & **name** (const **dds::core::string** &value)

  Sets the name.
- int32_t **ordinal** () const

  Gets the ordinal.
- **EnumMember** & **ordinal** (int32_t the_ordinal)

  Sets the ordinal.

8.112.1 Detailed Description

<<***value-type***>> *(*p. *350)* Represents a **EnumType** (p. 1535) member

Encapsulates the name and value (ordinal) of an IDL `enum` type

8.112.2 Constructor & Destructor Documentation

8.112.2.1 EnumMember()

```
dds::core::xtypes::EnumMember::EnumMember (
            const std::string & name,
            int32_t ordinal )
```

Creates a **EnumMember** (p. 1533) with a name and value (ordinal)

8.112.3 Member Function Documentation

8.112.3.1 name() [1/3]

```
const  dds::core::string & dds::core::xtypes::EnumMember::name ( ) const
```

Gets the name.

8.112.3.2 name() [2/3]

```
 dds::core::string & dds::core::xtypes::EnumMember::name ( )
```

Gets the member name.

8.112.3.3 name() [3/3]

```
 EnumMember & dds::core::xtypes::EnumMember::name (
            const  dds::core::string & value )
```

Sets the name.

8.112.3.4 ordinal() [1/2]

```
int32_t dds::core::xtypes::EnumMember::ordinal ( ) const
```

Gets the ordinal.

8.112.3.5 ordinal() [2/2]

```
EnumMember & dds::core::xtypes::EnumMember::ordinal (
            int32_t the_ordinal )
```

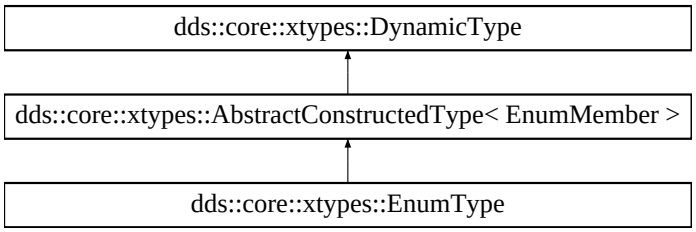
Sets the ordinal.

8.113 dds::core::xtypes::EnumType Class Reference

<<***value-type***>> *(*p. *350)* Represents an IDL `enum` type

`#include <dds/core/xtypes/EnumType.hpp>`

Inheritance diagram for dds::core::xtypes::EnumType:



Public Member Functions

- **EnumType** (const std::string & **name**)
  Creates an empty enum type.
- template<typename Container >
  EnumType (const std::string &the_name, const Container &the_members)
  Creates an enum with the members in a container.
- template<typename MemberIter >
  EnumType (const std::string &the_name, MemberIter begin, MemberIter end)
  Creates a enum with the members in an iterator range.
- **EnumType** (const std::string &the_name, std::initializer_list< **EnumMember** > the_members)
  Creates an enum with the members in an initializer_list.
- **MemberIndex find_member_by_ordinal** (int32_t ordinal) const
  Gets the index of the member with this ordinal value.
- **EnumType** & **add_member** (const **EnumMember** & **member**)
  Adds a member at the end.
- template<typename Container >
  EnumType & **add_members** (const Container &the_members)
  Adds all the members of a container at the end.
- **EnumType** & **add_members** (std::initializer_list< **EnumMember** > the_members)
  Adds all the members of an initializer_list at the end.
- template<typename MemberIter >
  EnumType & **add_members** (MemberIter begin, MemberIter end)
  Adds all the members in an iterator range at the end.

Additional Inherited Members

8.113.1 Detailed Description

<<***value-type***>> *(p. 350)* Represents an IDL `enum` type

Examples

Foo.hpp.

8.113.2 Constructor & Destructor Documentation

8.113.2.1 EnumType() [1/4]

```
dds::core::xtypes::EnumType::EnumType (
            const std::string & name )
```

Creates an empty enum type.

Members can be added after creation.

Parameters

name	The name of the type

8.113.2.2 EnumType() [2/4]

```
template<typename Container >
dds::core::xtypes::EnumType::EnumType (
            const std::string & the_name,
            const Container & the_members )  [inline]
```

Creates an enum with the members in a container.

Template Parameters

Container	A container that provides the member functions **begin()** (p. 1669) and **end()** (p. 1670) to iterate over **EnumMember** (p. 1533) elements.

Parameters

the_name	The name of the type
the_members	A container with the members for this enum type

8.113.2.3 EnumType() [3/4]

```
template<typename MemberIter >
dds::core::xtypes::EnumType::EnumType (
            const std::string & the_name,
            MemberIter begin,
            MemberIter end )  [inline]
```

Creates a enum with the members in an iterator range.

Template Parameters

MemberIter	A forward iterator of **EnumMember** (p. 1533) elements

Parameters

the_name	The name of the type
begin	The beginning of the range of EnumMembers
end	The end of the range of EnumMembers

8.113.2.4 EnumType() [4/4]

```
dds::core::xtypes::EnumType::EnumType (
            const std::string & the_name,
            std::initializer_list<  EnumMember > the_members )
```

Creates an enum with the members in an initializer_list.

Parameters

the_name	The name of the type
the_members	An ininitalizer_list of EnumMembers

8.113.3 Member Function Documentation

8.113.3.1 find_member_by_ordinal()

```
MemberIndex dds::core::xtypes::EnumType::find_member_by_ordinal (
            int32_t ordinal ) const
```

Gets the index of the member with this ordinal value.

Returns

The index (which can be passed to member(uint32_t) of the member with this ordinal value or INVALID_INDEX if that ordinal doesn't exist

8.113.3.2 add_member()

```
EnumType & dds::core::xtypes::EnumType::add_member (
            const  EnumMember & member )
```

Adds a member at the end.

8.113.3.3 add_members() [1/3]

```
template<typename Container >
EnumType & dds::core::xtypes::EnumType::add_members (
            const Container & the_members )  [inline]
```

Adds all the members of a container at the end.

8.113.3.4 add_members() [2/3]

```
EnumType & dds::core::xtypes::EnumType::add_members (
            std::initializer_list<  EnumMember > the_members )
```

Adds all the members of an initializer_list at the end.

8.113.3.5 add_members() [3/3]

```
template<typename MemberIter >
EnumType & dds::core::xtypes::EnumType::add_members (
            MemberIter begin,
            MemberIter end )  [inline]
```

Adds all the members in an iterator range at the end.

8.114 rti::test::EnvVarToken Class Reference

`#include <utilities.hpp>`

8.114.1 Detailed Description

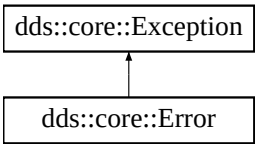
This class is used to create or modify an environment variable. If the variable didn't exist previously, the environment variable will be set and it will last as long as the life of this instance. If the variable already existed, this class will change its value to the provided one during the lifetime of this instance. After destruction, it will return to its previous value.

8.115 dds::core::Error Class Reference

A generic, unspecified **Error** (p. 1539).

`#include <Exception.hpp>`

Inheritance diagram for dds::core::Error:



Public Member Functions

- **Error** ()

  *Create an **Error** (p. 1539) exception with no message.*
- **Error** (const std::string &msg)

  *Create an **Error** (p. 1539) exception with a message.*
- virtual const char * **what** () const throw ()

  *Access the message contained in this **Error** (p. 1539) exception.*

8.115.1 Detailed Description

A generic, unspecified **Error** (p. 1539).

Inherits also from `std::exception`

8.115.2 Constructor & Destructor Documentation

8.115.2.1 Error() [1/2]

```
dds::core::Error::Error ( )
```

Create an **Error** (p. 1539) exception with no message.

8.115.2.2 Error() [2/2]

```
dds::core::Error::Error (
            const std::string & msg )  [explicit]
```

Create an **Error** (p. 1539) exception with a message.

Parameters

msg	The message to create the **Error** (p. 1539) with.

8.115.3 Member Function Documentation

8.115.3.1 what()

```
virtual const char * dds::core::Error::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **Error** (p. 1539) exception.

Returns

The message.

Implements **dds::core::Exception** (p. 1547).

8.116 rti::core::policy::Event Class Reference

<<***extension***>> *(*p. *116)* Configures the thread in a DomainParticipant that handles timed events.

`#include <rti/core/policy/CorePolicy.hpp>`

Public Member Functions

- **Event** ()
  Creates the default policy.
- **Event** (const **rti::core::ThreadSettings** &the_thread, int32_t the_initial_count, int32_t the_max_count)
  Creates an instance with all the parameters set.
- **Event** & **thread** (const **rti::core::ThreadSettings** &the_thread)
  <<***extension***>> *(*p. *116)* ***Event*** *(*p. *1541) thread QoS.*
- const **rti::core::ThreadSettings** & **thread** () const
  Getter (see setter with the same name)
- **rti::core::ThreadSettings** & **thread** ()
  Getter (see setter with the same name)
- **Event** & **initial_count** (int32_t the_initial_count)
  <<***extension***>> *(*p. *116) The initial number of events.*
- int32_t **initial_count** () const
  Getter (see setter with the same name)
- **Event** & **max_count** (int32_t the_max_count)
  <<***extension***>> *(*p. *116) The maximum number of events.*
- int32_t **max_count** () const
  Getter (see setter with the same name)

8.116.1 Detailed Description

<<***extension***>> *(*p. *116)* Configures the thread in a DomainParticipant that handles timed events.

In a **dds::domain::DomainParticipant** (p. 1326), a thread is dedicated to handle all timed events, including checking for timeouts and deadlines and executing internal and user-defined timeout or exception handling routines/callbacks.

This QoS policy allows you to configure thread properties such as priority level and stack size. You can also configure the maximum number of events that can be posted to the event thread. By default, a **dds::domain::DomainParticipant** (p. 1326) will dynamically allocate memory as needed for events posted to the event thread. However, by setting a maximum value or setting the initial and maximum value to be the same, you can either bound the amount of memory allocated for the event thread or prevent a **dds::domain::DomainParticipant** (p. 1326) from dynamically allocating memory for the event thread after initialization.

This QoS policy is an extension to the DDS standard.

Entity:

dds::domain::DomainParticipant (p. 1326)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.116.2 Constructor & Destructor Documentation

8.116.2.1 Event() [1/2]

```
rti::core::policy::Event::Event ( )  [inline]
```

Creates the default policy.

8.116.2.2 Event() [2/2]

```
rti::core::policy::Event::Event (
            const  rti::core::ThreadSettings & the_thread,
            int32_t the_initial_count,
            int32_t the_max_count )
```

Creates an instance with all the parameters set.

8.116.3 Member Function Documentation

8.116.3.1 thread() [1/3]

```
Event & rti::core::policy::Event::thread (
            const  rti::core::ThreadSettings & the_thread )
```

<<***extension***>> *(p. 116)* **Event** (p. 1541) thread QoS.

There is only one event thread.

Priority:
[default] The actual value depends on your architecture:
For Windows: -2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`.

Stack Size:
[default] The actual value depends on your architecture:
For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

Mask:
[default] mask = **rti::core::ThreadSettingsKindMask::floating_point()** (p. 2456) | **rti::core::ThreadSettingsKind↩ Mask::stdio()** (p. 2456)

Categories

Immutable (p. 243), **Concurrency** (p. 164)

8.116.3.2 thread() [2/3]

```
const rti::core::ThreadSettings & rti::core::policy::Event::thread ( ) const
```

Getter (see setter with the same name)

8.116.3.3 thread() [3/3]

```
rti::core::ThreadSettings & rti::core::policy::Event::thread ( )
```

Getter (see setter with the same name)

8.116.3.4 initial_count() [1/2]

```
Event & rti::core::policy::Event::initial_count (
            int32_t the_initial_count )
```

<<***extension***>> *(*p. *116)* The initial number of events.

[default] 256

[range] [1, 1 million], <= max_count

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

References **rti::core::policy::AsynchronousPublisher::disable_asynchronous_batch()**, and **rti::core::policy::↩ AsynchronousPublisher::disable_asynchronous_write()**.

8.116.3.5 initial_count() [2/2]

```
int32_t rti::core::policy::Event::initial_count ( ) const
```

Getter (see setter with the same name)

8.116.3.6 max_count() [1/2]

```
Event & rti::core::policy::Event::max_count (
            int32_t the_max_count )
```

<<***extension***>> *(p. 116)* The maximum number of events.

The maximum number of events. If the limit is reached, no new event can be added.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_count

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.116.3.7 max_count() [2/2]

```
int32_t rti::core::policy::Event::max_count ( ) const
```

Getter (see setter with the same name)

8.117 rti::core::status::EventCount< IntegerType > Class Template Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Encapsulates an event count containing a total count and an incremental count since the last time a status was read.

```
#include <Status.hpp>
```

Inherits std::pair< IntegerType, IntegerType >.

Public Member Functions

- IntegerType **total** () const
  The total count.
- IntegerType **change** () const
  The incremental count.

Related Functions

(Note that these are not member functions.)

- template<typename IntegerType >
  std::ostream & **operator**<< (std::ostream &out, const **EventCount**< IntegerType > &event_count)
  <<***extension***>> *(*p. *354) Prints an* ***rti::core::status::EventCount*** *(*p. *1544) to an output stream.*

8.117.1 Detailed Description

template<**typename IntegerType**>
class rti::core::status::EventCount< **IntegerType** >

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Encapsulates an event count containing a total count and an incremental count since the last time a status was read.

Template Parameters

IntegerType	The integer type used to measure the event counts.

8.117.2 Member Function Documentation

8.117.2.1 total()

```
template<typename IntegerType >
IntegerType  rti::core::status::EventCount< IntegerType >::total ( ) const  [inline]
```

The total count.

8.117.2.2 change()

```
template<typename IntegerType >
IntegerType  rti::core::status::EventCount< IntegerType >::change ( ) const  [inline]
```

The incremental count.

8.117.3 Friends And Related Function Documentation

8.117.3.1 operator<<()

```
template<typename IntegerType >
std::ostream & operator<< (
            std::ostream & out,
            const  EventCount< IntegerType > & event_count )  [related]
```

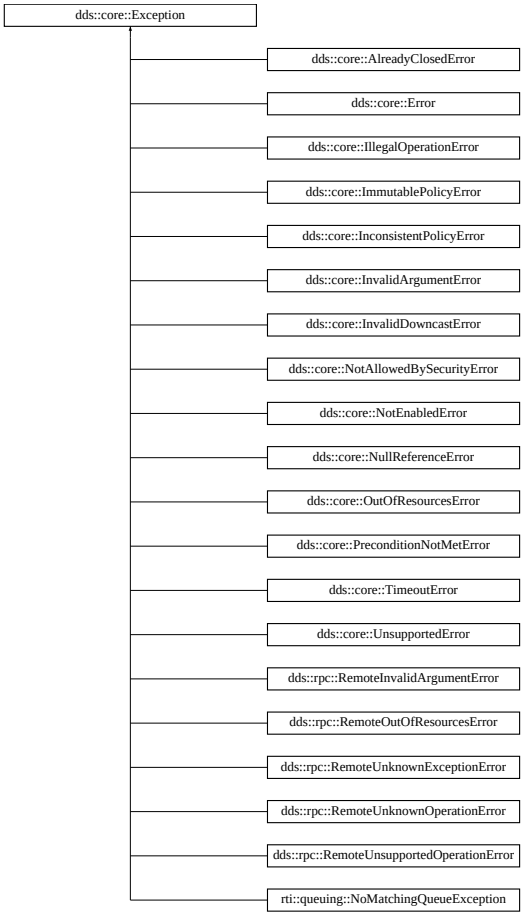
<<***extension***>> *(*p. *354)* Prints an **rti::core::status::EventCount** (p. 1544) to an output stream.

8.118 dds::core::Exception Class Reference

The abstract base class for all of the DDS exceptions which may be thrown by the API.

```
#include <Exception.hpp>
```

Inheritance diagram for dds::core::Exception:



Public Member Functions

- virtual const char * **what** () const =0 throw ()

  Get a description of the error.

8.118.1 Detailed Description

The abstract base class for all of the DDS exceptions which may be thrown by the API.

8.118.2 Member Function Documentation

8.118.2.1 what()

```
virtual const char * dds::core::Exception::what ( ) const throw ( )   [pure virtual]
```

Get a description of the error.

Implemented in **dds::core::Error** (p. 1540), **dds::core::AlreadyClosedError** (p. 806), **dds::core::IllegalOperationError** (p. 1608), **dds::core::NotAllowedBySecurityError** (p. 1868), **dds::core::ImmutablePolicyError** (p. 1609), **dds::core::InconsistentPolicyError** (p. 1610), **dds::core::InvalidArgumentError** (p. 1619), **dds::core::NotEnabledError** (p. 1869), **dds::core::OutOfResourcesError** (p. 1899), **dds::core::PreconditionNotMetError** (p. 1942), **dds::core::TimeoutError** (p. 2473), **dds::core::UnsupportedError** (p. 2590), **dds::core::InvalidDowncastError** (p. 1620), **dds::core::NullReferenceError** (p. 1870), **rti::queuing::NoMatchingQueueException** (p. 1836), **dds::rpc::RemoteUnknownOperationError** (p. 2167), **dds::rpc::RemoteUnsupportedOperationError** (p. 2168), **dds::rpc::RemoteInvalidArgumentError** (p. 2162), **dds::rpc::RemoteOutOfResourcesError** (p. 2163), and **dds::rpc::RemoteUnknownExceptionError** (p. 2166).

8.119 rti::topic::ExpressionProperty Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Provides additional information about the filter expression passed to the writer_compile method of **rti::topic::WriterContentFilter** (p. 2652)

```
#include <rti/topic/ContentFilter.hpp>
```

Public Member Functions

- **ExpressionProperty** ()

  *Create a default **ExpressionProperty** (p. 1547) with key_only_filter = false and writer_side_filter_optimization = false.*
- **ExpressionProperty** (bool the_key_only_filter, bool the_writer_side_filter_optimization)

  *Create an **ExpressionProperty** (p. 1547) with the provided key_only_filter, and writer_side_filter_optimization.*
- bool **key_only_filter** () const

  Get the value of key_only_filter.
- **ExpressionProperty** & **key_only_filter** (bool the_key_only_filter)

  Set the value for key_only_filter, indicating if the filter expression is based only on key fields. In this case, RTI Connext itself can cache the filtering results.
- bool **writer_side_filter_optimization** () const

  Get the value of writer_side_filter_optimization.
- **ExpressionProperty** & **writer_side_filter_optimization** (bool the_writer_side_filter_optimization)

  Set the value for writer_side_filter_optimization, indicating if the filter implementation can cache the filtering result for the provided expression.

8.119.1 Detailed Description

*<<**extension**>>* *(p. 354)* *<<**value-type**>>* *(p. 350)* Provides additional information about the filter expression passed to the writer_compile method of **rti::topic::WriterContentFilter** (p. 2652)

It is used by the filter implementation to indicate to the middleware whether or not the **WriterContentFilter** (p. 2652) will cache the result of filter evaluation.

See also

rti::topic::WriterContentFilter (p. 2652)

8.119.2 Constructor & Destructor Documentation

8.119.2.1 ExpressionProperty() [1/2]

```
rti::topic::ExpressionProperty::ExpressionProperty ( )  [inline]
```

Create a default **ExpressionProperty** (p. 1547) with key_only_filter = false and writer_side_filter_optimization = false.

8.119.2.2 ExpressionProperty() [2/2]

```
rti::topic::ExpressionProperty::ExpressionProperty (
            bool the_key_only_filter,
            bool the_writer_side_filter_optimization )  [inline]
```

Create an **ExpressionProperty** (p. 1547) with the provided key_only_filter, and writer_side_filter_optimization.

Parameters

the_key_only_filter	The value for key_only_filter
the_writer_side_filter_optimization	The value for writer_side_filter_optimization

8.119.3 Member Function Documentation

8.119.3.1 key_only_filter() [1/2]

```
bool rti::topic::ExpressionProperty::key_only_filter ( ) const  [inline]
```

Get the value of key_only_filter.

See also

key_only_filter(bool the_key_only_filter) (p. 1549)

8.119.3.2 key_only_filter() [2/2]

```
ExpressionProperty & rti::topic::ExpressionProperty::key_only_filter (
            bool the_key_only_filter )  [inline]
```

Set the value for key_only_filter, indicating if the filter expression is based only on key fields. In this case, RTI Connext itself can cache the filtering results.

When this field is set to true, it indicates to RTI Connext that the filter expression is based only on key fields.

Parameters

the_key_only_filter	The value to set for key_only_filter

8.119.3.3 writer_side_filter_optimization() [1/2]

```
bool rti::topic::ExpressionProperty::writer_side_filter_optimization ( ) const  [inline]
```

Get the value of writer_side_filter_optimization.

See also

writer_side_filter_optimization(bool the_writer_side_filter_optimization) (p. 1549)

8.119.3.4 writer_side_filter_optimization() [2/2]

```
ExpressionProperty & rti::topic::ExpressionProperty::writer_side_filter_optimization (
            bool the_writer_side_filter_optimization )  [inline]
```

Set the value for writer_side_filter_optimization, indicating if the filter implementation can cache the filtering result for the provided expression.

When this field is set to true, RTI Connext will do no caching or explicit filter evaluation for the associated **dds::sub::↩ DataReader** (p. 976). Instead, it will rely on the filter implementation to provide appropriate results.

Parameters

the_writer_side_filter_optimization	The value to set for writer_side_filter_optimization

8.120 rti::topic::extensibility< TopicType > Struct Template Reference

<<***extension***>> *(p. 354)* Indicates the extensibility kind of a topic-type

`#include <rti/topic/TopicTraits.hpp>`

8.120.1 Detailed Description

template<typename TopicType>
struct rti::topic::extensibility< TopicType >

<<***extension***>> *(p. 354)* Indicates the extensibility kind of a topic-type

This trait type is specialized for **IDL-generated types** (p. 606) provide a member `kind` of type **dds::core::xtypes::ExtensibilityKind** (p. 633).

For example, given the following IDL type, **Foo** (p. 1588):

```
struct Foo {
    long x;
}; //@Extensibility MUTABLE_EXTENSIBILITY
```

You can obtain its extensility kind:

```
assert (rti::topic::extensibility<Foo>::kind
     == dds::core::xtypes::ExtensibilityKind::MUTABLE);
```

8.121 dds::core::xtypes::ExtensibilityKind_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) ExtensibilityKind.

`#include <Annotations.hpp>`

Public Types

- enum **type** {
  FINAL ,
  EXTENSIBLE ,
  MUTABLE }

8.121.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) ExtensibilityKind.

8.121.2 Member Enumeration Documentation

8.121.2.1 type

enum **dds::core::xtypes::ExtensibilityKind_def::type**

The underlying enum type

Enumerator

FINAL	Specifies that a type has FINAL extensibility. A type may be final, indicating that the range of its possible values is strictly defined. In particular, it is not possible to add elements to members of collection or aggregated types while maintaining type assignability. The following types are always final: • **dds::core::xtypes::TypeKind_def::ARRAY_TYPE** (p. 2573) • All primitive types
EXTENSIBLE	Specifies that a type has EXTENSIBLE extensibility. A type may be appendable (formerly called extensible), indicating that two types, where one contains all of the elements/members of the other plus additional elements/members appended to the end, may remain assignable.
MUTABLE	Specifies that a type has MUTABLE extensibility. A type may be mutable, indicating that two types may differ from one another in the additional, removal, and/or transposition of elements/members while remaining assignable. The following types are always mutable: • **dds::core::xtypes::TypeKind_def::SEQUENCE_TYPE** (p. 2573) • **dds::core::xtypes::TypeKind_def::STRING_TYPE** (p. 2574) • **dds::core::xtypes::TypeKind_def::WSTRING_TYPE** (p. 2574)

8.122 dds::core::external< T > Class Template Reference

A managed reference to an object.

`#include <dds/core/External.hpp>`

Public Types

- using **shared_ptr** = std::shared_ptr< T >

  The smart pointer that external<T> uses.

Public Member Functions

- **external** () OMG_NOEXCEPT

  Creates an empty shared object.
- **external** (T *p, bool locked=false)

  Creates a new shared object with the managed object `p`.
- **external** (**shared_ptr** p)

  Creates an `external` instance that shares the managed object and reference counting with an existing `shared_ptr`.
- **external** (const **external** &other)

  Creates a shared object from an existing shared object.
- ~**external** ()

  Destroys the managed object if this is the last shared object pointing to it.
- **external** & **operator=** (const **external** &other)

  Assigns another external, depending on the locked state.
- T & **operator*** () OMG_NOEXCEPT

  Obtains a reference to the managed object.
- const T & **operator*** () const OMG_NOEXCEPT

  Obtains a const reference to the managed object.
- T * **get** () OMG_NOEXCEPT

  Obtains a pointer to the managed object.
- const T * **get** () const OMG_NOEXCEPT

  Obtains a const pointer to the managed object.
- **shared_ptr** **get_shared_ptr** ()

  Obtains a `shared_ptr` to the managed object.
- T * **operator->** () OMG_NOEXCEPT

  Allows accessing members of the managed object.
- const T * **operator->** () const OMG_NOEXCEPT

  Allows accessing members of the managed object.
- bool **operator==** (const **external**< T > &other) const OMG_NOEXCEPT

  Returns whether two shared objects manage the same object or are both empty.
- bool **operator!=** (const **external**< T > &other) const

  Returns whether two shared objects do not manage the same object.
- RTI_EXPLICIT_CONVERSION **operator bool** () const OMG_NOEXCEPT

  Checks if there is a managed object (is not null) or not (is null)
- bool **is_locked** () const

  Indicates whether the managed object is locked or not.

Friends

- void **swap** (**external**< T > &a, **external**< T > &b) OMG_NOEXCEPT

  Swaps the managed objects and the locked state.

8.122.1 Detailed Description

template<**typename T**>
class dds::core::external< **T** >

A managed reference to an object.

Template Parameters

T	The type of the object this external<T> manages

Members of an **IDL** (p. 606) type marked with the @Shared or @external tag map to this C++ type. External members can share the same object when publishing different data samples. This can make more efficient the publication of large objects by saving extra copies.

This class behaves like a std::shared_ptr, except when the object is in "locked" state.

Objects of this class created with any of the constructors documented here behave like a std::shared_ptr<T>. Objects that the middleware returns in a LoanedSample (for example, from **dds::sub::DataReader::take** (p. 997)) are in a "locked" state and its contents can't be shared or modified. When assigning a sample with locked external members into another one where those members are not locked, the contents are copied, not shared. Modifying a locked object is not allowed. This special behavior prevents applications from corrupting loaned memory.

See also

Working with IDL types (p. 606)

8.122.2 Constructor & Destructor Documentation

8.122.2.1 external() [1/4]

```
template<typename T >
dds::core::external< T >::external ( )  [inline]
```

Creates an empty shared object.

8.122.2.2 external() [2/4]

```
template<typename T >
dds::core::external< T >::external (
            T * p,
            bool locked = false )  [inline]
```

Creates a new shared object with the managed object p.

Parameters

p	The pointer to manage
locked	Should always be false except in objects created internally by the middleware.

8.122.2.3 external() [3/4]

```
template<typename T >
dds::core::external< T >::external (
              shared_ptr p )  [inline]
```

Creates an external instance that shares the managed object and reference counting with an existing shared_ptr.

This constructor is implicit to allow the direct usage of shared_ptr.

For example:

```
shared_ptr<MyResource> resource(new MyResource);
// pass resource directly to a data-type with a my_resource external
// member whose setter expects an external<MyResource>.
sample1.my_resource(resource);
sample2.my_resource(resource);
// ...
writer.write(sample1);
writer.write(sample2);
```

8.122.2.4 external() [4/4]

```
template<typename T >
dds::core::external< T >::external (
             const  external< T > & other )  [inline]
```

Creates a shared object from an existing shared object.

In general, the new shared object shares ownership with other. Only if other.is_locked(), then the new shared object is allocated and its contents copied from other.

Postcondition

if !other.**is_locked()** (p. 1557), then *this == other. If other.is_locked, then *(*this) == *other but *this != other.

8.122.2.5 ~external()

```
template<typename T >
dds::core::external< T >::~ external ( )  [inline]
```

Destroys the managed object if this is the last shared object pointing to it.

8.122.3 Member Function Documentation

8.122.3.1 operator=()

```
template<typename T >
external &  dds::core::external< T >::operator= (
             const  external< T > & other )  [inline]
```

Assigns another external, depending on the locked state.

Depending on whether `this` or `other` are locked the behavior varies:

- If `this->`**is_locked()** (p. 1557) this operation throws **dds::core::PreconditionNotMet↩ Error** (p. 1941)
- If `other.is_locked()` it performs a deep copy of `*other`
- In any other case, after the assignment, `*this` and `other` share the same reference to the underlying object, that is, a `shared_ptr` assignment.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if this->**is_locked()** (p. 1557)

References **dds::core::external< T >::is_locked()**.

8.122.3.2 operator*() [1/2]

```
template<typename T >
T &  dds::core::external< T >::operator* ( )  [inline]
```

Obtains a reference to the managed object.

8.122.3.3 operator*() [2/2]

```
template<typename T >
const T &  dds::core::external< T >::operator* ( ) const  [inline]
```

Obtains a const reference to the managed object.

8.122.3.4 get() [1/2]

```
template<typename T >
T * dds::core::external< T >::get ( ) [inline]
```

Obtains a pointer to the managed object.

Warning

The returned pointer may become invalid if this is destroyed

8.122.3.5 get() [2/2]

```
template<typename T >
const T * dds::core::external< T >::get ( ) const [inline]
```

Obtains a const pointer to the managed object.

Warning

The returned pointer may become invalid if this is destroyed

8.122.3.6 get_shared_ptr()

```
template<typename T >
shared_ptr dds::core::external< T >::get_shared_ptr ( ) [inline]
```

Obtains a `shared_ptr` to the managed object.

The returned shared_ptr shares the same reference count.

Exceptions

dds::core::IllegalOperationError *(p. 1608)*	if **`is_locked()`** (p. 1557).

8.122.3.7 operator->() [1/2]

```
template<typename T >
T * dds::core::external< T >::operator-> ( ) [inline]
```

Allows accessing members of the managed object.

8.122.3.8 operator->() [2/2]

```
template<typename T >
const T *  dds::core::external< T >::operator-> ( ) const  [inline]
```

Allows accessing members of the managed object.

8.122.3.9 operator==()

```
template<typename T >
bool  dds::core::external< T >::operator== (
            const  external< T > & other ) const  [inline]
```

Returns whether two shared objects manage the same object or are both empty.

8.122.3.10 operator"!=()

```
template<typename T >
bool  dds::core::external< T >::operator!= (
            const  external< T > & other ) const  [inline]
```

Returns whether two shared objects do not manage the same object.

8.122.3.11 operator bool()

```
template<typename T >
RTI_EXPLICIT_CONVERSION  dds::core::external< T >::operator bool ( ) const  [inline]
```

Checks if there is a managed object (is not null) or not (is null)

8.122.3.12 is_locked()

```
template<typename T >
bool  dds::core::external< T >::is_locked ( ) const  [inline]
```

Indicates whether the managed object is locked or not.

Referenced by **dds::core::external< T >::operator=()**.

8.122.4 Friends And Related Function Documentation

8.122.4.1 swap

```
template<typename T >
void swap (
              external< T > & a,
              external< T > & b )  [friend]
```

Swaps the managed objects and the locked state.

8.123 dds::topic::Filter Class Reference

Defines the filter to create a **ContentFilteredTopic** (p. 954).

```
#include <dds/topic/Filter.hpp>
```

Public Member Functions

- **Filter** (const std::string &filter_expression)

  Creates a filter with a expression with no parameters.
- template<typename FwdIterator >
  Filter (const std::string &filter_expression, const FwdIterator &params_begin, const FwdIterator &params_end)

  Creates a filter with an expression that contains parameters.
- **Filter** (const std::string &query_expression, const std::vector< std::string > &params)

  Creates a filter with an expression that contains parameters.
- const std::string & **expression** () const

  Gets the filter expression.
- const_iterator **begin** () const

  Provides the begin iterator to the parameter list.
- const_iterator **end** () const

  The end iterator to the parameter list.
- iterator **begin** ()

Provides the begin iterator to the parameter list.

- iterator **end** ()

  The end iterator to the parameter list.

- template<typename FwdITerator >
  Filter & **parameters** (const FwdITerator &the_begin, const FwdITerator the_end)

  Modifies the parameters.

- **Filter** & **add_parameter** (const std::string &param)

  Appends a parameter.

- size_t **parameters_length** () const

  Gets the number of parameters.

- std::string **name** () const

  <<***extension***>> (p. *354) Gets the filter name*

- **dds::topic::Filter** & **name** (const std::string &the_name)

  <<***extension***>> (p. *354) Sets a filter name*

8.123.1 Detailed Description

Defines the filter to create a **ContentFilteredTopic** (p. 954).

Contains the filter expression and optionally the expression parameters. It also contains the filter name, which allows selecting the filter class to use–by default it's the built-in SQL filter.

8.123.2 Constructor & Destructor Documentation

8.123.2.1 Filter() [1/3]

```
dds::topic::Filter::Filter (
            const std::string & filter_expression )  [inline]
```

Creates a filter with a expression with no parameters.

See also

Queries and Filters Syntax (p. 84)

8.123.2.2 Filter() [2/3]

```
template<typename FwdIterator >
dds::topic::Filter::Filter (
            const std::string & filter_expression,
            const FwdIterator & params_begin,
            const FwdIterator & params_end )  [inline]
```

Creates a filter with an expression that contains parameters.

Template Parameters

FwdIterator	A forwar iterator whose value type is std::string (or convertible to)

Parameters

filter_expression	The filter expression
params_begin	The beginning of the range of expression parameters
params_end	The end of the range (the number of parameter to create a **ContentFilteredTopic** (p. 954) can't exceed 100)

See also

Queries and Filters Syntax (p. 84)

8.123.2.3 Filter() [3/3]

```
dds::topic::Filter::Filter (
            const std::string & query_expression,
            const std::vector< std::string > & params )  [inline]
```

Creates a filter with an expression that contains parameters.

Parameters

query_expression	The query expression
params	The expression parameters (the number of parameter to create a **ContentFilteredTopic** (p. 954) can't exceed 100)

See also

Queries and Filters Syntax (p. 84)

8.123.3 Member Function Documentation

8.123.3.1 expression()

```
const std::string & dds::topic::Filter::expression ( ) const  [inline]
```

Gets the filter expression.

8.123.3.2 begin() [1/2]

```
const_iterator dds::topic::Filter::begin ( ) const  [inline]
```

Provides the begin iterator to the parameter list.

8.123.3.3 end() [1/2]

```
const_iterator dds::topic::Filter::end ( ) const  [inline]
```

The end iterator to the parameter list.

8.123.3.4 begin() [2/2]

```
iterator dds::topic::Filter::begin ( )  [inline]
```

Provides the begin iterator to the parameter list.

8.123.3.5 end() [2/2]

```
iterator dds::topic::Filter::end ( )  [inline]
```

The end iterator to the parameter list.

8.123.3.6 parameters()

```
template<typename FwdITerator >
Filter & dds::topic::Filter::parameters (
            const FwdITerator & the_begin,
            const FwdITerator the_end )  [inline]
```

Modifies the parameters.

8.123.3.7 add_parameter()

```
Filter & dds::topic::Filter::add_parameter (
            const std::string & param )  [inline]
```

Appends a parameter.

8.123.3.8 parameters_length()

```
size_t dds::topic::Filter::parameters_length ( ) const  [inline]
```

Gets the number of parameters.

8.123.3.9 name() [1/2]

```
std::string name ( ) const
```

<<***extension***>> *(*p. *354)* Gets the filter name

Returns

The name of the filter, or an empty string when using the default SQL filter.

8.123.3.10 name() [2/2]

```
dds::topic::Filter & name (
            const std::string & the_name )
```

<<***extension***>> *(*p. *354)* Sets a filter name

You can use one of the built-in filters (**rti::topic::sql_filter_name** (p. 49) and **rti::topic::stringmatch_filter_name** (p. 49)), or a custom filter, which you need to register using **dds::domain::DomainParticipant::register_contentfilter()** (p. 1349).

[default] Empty string (equivalent to **rti::topic::sql_filter_name** (p. 49))

8.124 rti::topic::FilterSampleInfo Class Reference

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Provides meta information associated with the sample.

```
#include <rti/topic/FilterSampleInfo.hpp>
```

Public Member Functions

- **rti::core::SampleIdentity related_sample_identity** () const
  Get the related sample identity of the sample being filtered.
- **rti::core::Guid related_source_guid** () const
  Get the related source GUID of the sample being filtered.
- **rti::core::Guid related_reader_guid** () const
  Get the related reader GUID of the sample being filtered.
- **rti::core::SampleIdentity sample_identity** () const
  Get the sample identity of the sample being filtered.
- **rti::core::Guid source_guid** () const
  Get the source GUID of the sample being filtered.
- int32_t **priority** () const
  Get the priority of the sample being filtered.

8.124.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Provides meta information associated with the sample.

This can be used by a content filter to perform filtering on meta data.

8.124.2 Member Function Documentation

8.124.2.1 related_sample_identity()

```
rti::core::SampleIdentity rti::topic::FilterSampleInfo::related_sample_identity ( ) const  [inline]
```

Get the related sample identity of the sample being filtered.

The value of this field identifies another sample that is logically related to the one that is written. For example, the **dds::pub::DataWriter** (p. 1138) created by a Replier uses this field to associate the identity of the request sample with a response sample.

The related_sample_identity can be set using the field **rti::pub::WriteParams::related_sample_identity** (p. 2646) when writing a sample using the method **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179).

Default value **rti::core::SampleIdentity::unknown()** (p. 2284) denotes a sample not associated with any other sample.

A **dds::sub::DataReader** (p. 976) can inspect the related sample identity of a received sample by accessing the fields **dds::sub::SampleInfo::related_original_publication_virtual_guid** (p. 2292) and **dds::sub::SampleInfo::related_↩ original_publication_virtual_sequence_number** (p. 2292).

8.124.2.2 related_source_guid()

```
rti::core::Guid rti::topic::FilterSampleInfo::related_source_guid ( ) const  [inline]
```

Get the related source GUID of the sample being filtered.

The related_source_guid can be set by using the field **rti::pub::WriteParams::related_source_guid** (p. 2651) when writing a sample using the method **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179).

Default value **rti::core::Guid::unknown()** (p. 1597) denotes a sample as lacking a related data source.

A **dds::sub::DataReader** (p. 976) can inspect the related_source_guid of a received sample by accessing the field **dds::sub::SampleInfo::related_source_guid** (p. 2294).

8.124.2.3 related_reader_guid()

```
rti::core::Guid rti::topic::FilterSampleInfo::related_reader_guid ( ) const  [inline]
```

Get the related reader GUID of the sample being filtered.

The related_reader_guid can be set by using the field **rti::pub::WriteParams::related_reader_guid** (p. 2651) when writing a sample using the method **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179).

Default value **rti::core::Guid::unknown()** (p. 1597) denotes a sample without an associated DataReader.

A **dds::sub::DataReader** (p. 976) can inspect the related_reader_guid of a received sample by accessing the field **dds::sub::SampleInfo::related_source_guid** (p. 2294).

8.124.2.4 sample_identity()

```
rti::core::SampleIdentity rti::topic::FilterSampleInfo::sample_identity ( ) const  [inline]
```

Get the sample identity of the sample being filtered.

The value of this field uniquely identifies a sample within its source DataWriter.

The identity of a sample can be set by using the field **rti::pub::WriteParams::identity** (p. 2646) when writing a sample using the method **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179).

Default value **rti::core::SampleIdentity::unknown()** (p. 2284) can be used to let the DataWriter assign a default identity to a sample. In this case, sample_identity.writer_guid will be set to the GUID of the DataWriter, while sample_identity.↩ sequence_number will be set to the next sequence number to be written by the DataWriter.

A **dds::sub::DataReader** (p. 976) can inspect the identity of a received sample by accessing the fields **dds↩ ::sub::SampleInfo::original_publication_virtual_guid** (p. 2291) and **dds::sub::SampleInfo::original_publication↩ _virtual_sequence_number** (p. 2291).

8.124.2.5 source_guid()

```
rti::core::Guid rti::topic::FilterSampleInfo::source_guid ( ) const  [inline]
```

Get the source GUID of the sample being filtered.

The source_guid can be set by using the field **rti::pub::WriteParams::source_guid** (p. 2650) when writing a sample using the method **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179).

Default value **rti::core::Guid::unknown()** (p. 1597) denotes a sample without a logical data source associated with it.

A **dds::sub::DataReader** (p. 976) can inspect the logical data source of a sample by accessing the field **dds::sub::SampleInfo::source_guid** (p. 2293).

8.124.2.6 priority()

```
int32_t rti::topic::FilterSampleInfo::priority ( ) const  [inline]
```

Get the priority of the sample being filtered.

To use publication priorities, the **rti::core::policy::PublishMode** (p. 2013) must be set for asynchronous publishing and the **dds::pub::DataWriter** (p. 1138) must use a **rti::pub::FlowController** (p. 1572) with a FlowControllerSchedulingPolicy_def::HIGHEST_PRIORITY_FIRST

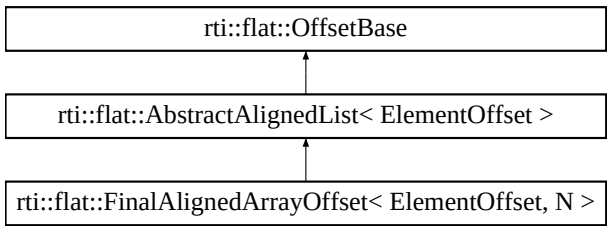
For Multi-channel DataWriters, the publication priority of a sample may be used as a filter criteria for determining channel membership.

8.125 rti::flat::FinalAlignedArrayOffset< ElementOffset, N > Class Template Reference

Offset to an array of final elements.

```
#include <SequenceOffsets.hpp>
```

Inheritance diagram for rti::flat::FinalAlignedArrayOffset< ElementOffset, N >:



Public Member Functions

- ElementOffset **get_element** (unsigned int i)

  Gets the Offset to an element.

Additional Inherited Members

8.125.1 Detailed Description

template<typename ElementOffset, unsigned int N>
class rti::flat::FinalAlignedArrayOffset< ElementOffset, N >

Offset to an array of final elements.

Template Parameters

ElementOffset	A final struct Offset, such as **MyFlatFinalOffset** (p. 1759).
N	The array bound. For multidimensional arrays, `N` is the product of each dimension bound.

Represents an Offset to an array member and allows getting an Offset to each of its elements.

The following code shows how to use a **FinalAlignedArrayOffset** (p. 1565) to initialize an array member with **MyFlat↩ MutableBuilder** (p. 1763):

```
MyFlatMutableBuilder builder = rti::flat::build_data(writer);
auto array_offset = builder.add_my_final_array();
for (MyFlatFinalOffset element_offset : array_offset) {
    element_offset.my_primitive(3);
    // ...
}
// Or access an element directly:
auto element_offset = array_offset.get_element(3);
element_offset.my_primitive(3);
```

A more efficient way to access a final array, provided it complies with the required preconditions, is through **rti::flat↩ ::plain_cast()** (p. 417).

FinalArrayOffset (p. 1567) and **FinalAlignedArrayOffset** (p. 1565) provide the same interface, but have different implementation details. **FinalArrayOffset** (p. 1567) is used when the array member is part of a final type too, whereas **FinalAlignedArrayOffset** (p. 1565) corresponds to an array inside a mutable type.

A **FinalAlignedArrayOffset** (p. 1565) may meet the requirements to be cast to an array of the equivalent plain C++ element type (see **rti::flat::plain_cast()** (p. 417)).

See also

MutableArrayOffset (p. 1755) encapsulates arrays of variable-size elements

8.125.2 Member Function Documentation

8.125.2.1 get_element()

```
template<typename ElementOffset , unsigned int N>
ElementOffset  rti::flat::FinalAlignedArrayOffset< ElementOffset, N >::get_element (
            unsigned int i )  [inline]
```

Gets the Offset to an element.

Parameters

i	The zero-based index to the element

Returns

The Offset to the element in the i-th position

See also

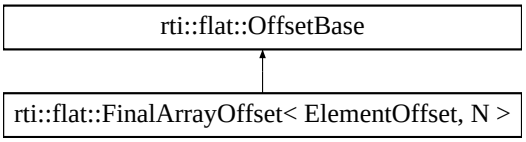
rti::flat::plain_cast() (p. 417) for a method to access **all** (p. 699) the elements at once

8.126 rti::flat::FinalArrayOffset< ElementOffset, N > Class Template Reference

Offset to an array of final elements.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::FinalArrayOffset< ElementOffset, N >:



Public Member Functions

- ElementOffset **get_element** (unsigned int i)

  Gets the Offset to an element.

8.126.1 Detailed Description

template<typename ElementOffset, unsigned int N>
class rti::flat::FinalArrayOffset< ElementOffset, N >

Offset to an array of final elements.

Template Parameters

ElementOffset	A final struct Offset, such as **MyFlatFinalOffset** (p. 1759).
N	The array bound. For multidimensional arrays, N is the product of each dimension bound.

Represents an Offset to an array member and allows getting an Offset to each of its elements.

FinalArrayOffset (p. 1567) and **FinalAlignedArrayOffset** (p. 1565) provide the same interface, but have different implementation details. **FinalArrayOffset** (p. 1567) is used when the array member is part of a final type too, whereas **FinalAlignedArrayOffset** (p. 1565) corresponds to an array inside a mutable type.

A **FinalArrayOffset** (p. 1567) may meet the requirements to be cast to an array of the equivalent plain C++ element type (see **rti::flat::plain_cast()** (p. 417)).

8.126.2 Member Function Documentation

8.126.2.1 get_element()

```
template<typename ElementOffset , unsigned int N>
ElementOffset  rti::flat::FinalArrayOffset< ElementOffset, N >::get_element (
             unsigned int i )  [inline]
```

Gets the Offset to an element.

Parameters

i	The zero-based index to the element

Returns

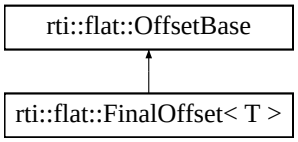
The Offset to the element in the i-th position

8.127 rti::flat::FinalOffset< T > Class Template Reference

The base class of all Offsets to a final struct type.

```
#include <Offset.hpp>
```

Inheritance diagram for rti::flat::FinalOffset< T >:



Additional Inherited Members

8.127.1 Detailed Description

template<typename T>
class rti::flat::FinalOffset< T >

The base class of all Offsets to a final struct type.

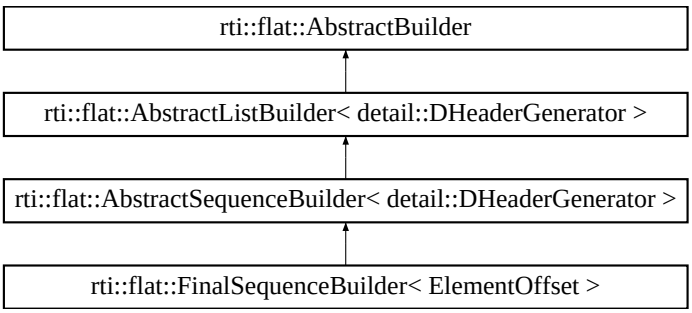
This class contains only implementation details; all the public accessors are defined in the generated type (**MyFlat↩ FinalOffset** (p. 1759)).

8.128 rti::flat::FinalSequenceBuilder< ElementOffset > Class Template Reference

Builds a sequence member of fixed-size elements.

`#include <SequenceBuilders.hpp>`

Inheritance diagram for rti::flat::FinalSequenceBuilder< ElementOffset >:



Public Member Functions

- ElementOffset **add_next** ()
  Adds the next element.
- **FinalSequenceBuilder** & **add_n** (unsigned int count)
  Adds a number of elements at once.
- **Offset finish** ()
  Finishes building the sequence.

Additional Inherited Members

8.128.1 Detailed Description

template<typename ElementOffset>
class rti::flat::FinalSequenceBuilder< ElementOffset >

Builds a sequence member of fixed-size elements.

Template Parameters

ElementOffset	The Offset type for the elements of the sequence

To add an element, call **add_next()** (p. 1570) and use the ElementOffset it returns to initialize the element's values. An empty sequence can be built by calling **finish()** (p. 1570) without any call to **add_next()** (p. 1570).

This class doesn't enforce the sequence bound set in IDL.

The following example uses a **FinalSequenceBuilder** (p. 1569) to initialize a sequence member of **MyFlatMutable↩ Builder** (p. 1763) with two elements:

```
MyFlatMutableBuilder builder = rti::flat::build_data(writer);
auto seq_builder = builder.build_my_final_seq();
MyFlatFinalOffset element = seq_builder.add_next();
element.my_primitive(1);
// ... continue initializing the first element
element = seq_builder.add_next();
element.my_primitive(2);
// ... continue initializing the second element

seq_builder.finish();
```

If the element type meets certain requirements, **rti::flat::plain_cast()** (p. 417) provides a more efficient way to initialize a sequence of final elements.

8.128.2 Member Function Documentation

8.128.2.1 add_next()

```
template<typename ElementOffset >
ElementOffset  rti::flat::FinalSequenceBuilder< ElementOffset >::add_next ( )  [inline]
```

Adds the next element.

Returns

The Offset that can be used to set the element values

8.128.2.2 add_n()

```
template<typename ElementOffset >
FinalSequenceBuilder &  rti::flat::FinalSequenceBuilder< ElementOffset >::add_n (
            unsigned int count )  [inline]
```

Adds a number of elements at once.

This is an alternative to **add_next()** (p. 1570).

To initialize the elements, call **finish()** (p. 1570) and use the Offset it returns to access the elements.

8.128.2.3 finish()

```
template<typename ElementOffset >
Offset rti::flat::FinalSequenceBuilder< ElementOffset >::finish ( ) [inline]
```

Finishes building the sequence.

Returns

An Offset to the member that has been built.

See also

discard() (p. 784)

8.129 rti::flat::flat_type_traits< T > Struct Template Reference

Given a **Sample** (p. 2274), an Offset or a Builder, it allows obtaining the other types.

8.129.1 Detailed Description

template<typename T>
struct rti::flat::flat_type_traits< T >

Given a **Sample** (p. 2274), an Offset or a Builder, it allows obtaining the other types.

Template Parameters

T	One of the following: • A **Sample** (p. 2274) type, such as **MyFlatMutable** (p. 413) • An Offset type, such as **MyFlatMutableOffset** (p. 1770) • A Builder type, such as **MyFlatMutableBuilder** (p. 1763)

Given `T`, this type provides the following typedefs:

- `flat_type_traits<T>::offset`, T's related offset type (undefined if T itself is an Offset)
- `flat_type_traits<T>::builder`, T's related builder type (undefined if T itself is a Builder, or the topic-type is not mutable)
- `flat_type_traits<T>::flat_type`, T's related **Sample** (p. 2274) type (undefined if T itself is a **Sample** (p. 2274) type)

- `flat_type_traits<T>::plain_type`, T's equivalent definition as a plain (non-FlatData) type.

For example, for `T` = **MyFlatMutable** (p. 413), **flat_type_traits** (p. 1571) is defined as follows:

```
template <>
 struct flat_type_traits<MyFlatMutable> {
     typedef MyFlatMutablePlainHelper plain_type;
     typedef MyFlatMutableOffset offset;
     typedef MyFlatMutableBuilder builder;
 };
```

Or if `T` = **MyFlatMutableOffset** (p. 1770):

```
template <>
struct flat_type_traits<MyFlatMutableOffset> {
      typedef MyFlatMutable flat_type;
      typedef MyFlatMutablePlainHelper plain_type;
      typedef MyFlatMutableBuilder builder;
};
```

See also

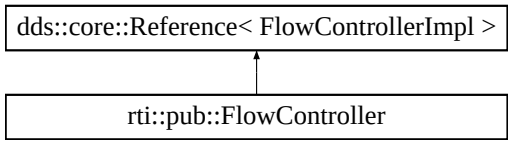
rti::flat::plain_cast() (p. 417)

8.130 rti::pub::FlowController Class Reference

*<<**extension**>> (p. 354) <<**reference-type**>> (p. 351)* A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **dds::pub::DataWriter** (p. 1138) instances are allowed to write data.

`#include <FlowController.hpp>`

Inheritance diagram for rti::pub::FlowController:



Public Member Functions

- **FlowController** (**dds::domain::DomainParticipant** the_participant, const std::string &the_name, const **FlowControllerProperty** &the_property= **FlowControllerProperty**())

  *Creates a **FlowController** (p. 1572) with specific properties.*
- std::string **name** () const

  *Gets the name of this **FlowController** (p. 1572).*
- **dds::domain::DomainParticipant participant** () const

  *Gets the participant associated to this **FlowController** (p. 1572).*
- **FlowControllerProperty property** () const

  *Gets the configuration of this **FlowController** (p. 1572).*
- void **property** (const **FlowControllerProperty** &prop) const

  *Gets the configuration of this **FlowController** (p. 1572).*

- void **trigger_flow** ()

  *Provides an external way to trigger a **FlowController** (p. 1572).*
- void **retain** ()

  Disables the automatic destruction of this object.
- void **close** ()

  Manually destroys this object.
- bool **closed** () const

  *Returns true if this **FlowController** (p. 1572) has been closed.*

Static Public Attributes

- static OMG_DDS_API_CLASS_VARIABLE const std::string **DEFAULT_NAME**

  *Name that identifies the built-in default **FlowController** (p. 1572).*
- static OMG_DDS_API_CLASS_VARIABLE const std::string **FIXED_RATE_NAME**

  *Name that identifies the built-in fixed-rate **FlowController** (p. 1572).*
- static OMG_DDS_API_CLASS_VARIABLE const std::string **ON_DEMAND_NAME**

  *Name that identifies the built-in on-demand **FlowController** (p. 1572).*

Related Functions

(Note that these are not member functions.)

- **FlowController find_flow_controller** (**dds::domain::DomainParticipant participant**, const std::string & **name**)

  *Retrieves an existing **FlowController** (p. 1572).*

8.130.1 Detailed Description

*<<**extension**>> (p. 354) <<**reference-type**>> (p. 351)* A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **dds::pub::DataWriter** (p. 1138) instances are allowed to write data.

Note

A **FlowController** (p. 1572) provides all the functions of a *<<**reference-type**>> (p. 351)*, including **close()** (p. 1576) and **retain()** (p. 1576).

8.130.2 Constructor & Destructor Documentation

8.130.2.1 FlowController()

```
rti::pub::FlowController::FlowController (
              dds::domain::DomainParticipant the_participant,
             const std::string & the_name,
             const  FlowControllerProperty & the_property =  FlowControllerProperty() )  [inline]
```

Creates a **FlowController** (p. 1572) with specific properties.

Parameters

the_participant	The DomainParticipant where this **FlowController** (p. 1572) exists
the_name	Name to refer to this **FlowController** (p. 1572)
the_property	Determines how to shape the network traffic

The created **rti::pub::FlowController** (p. 1572) is associated with a **dds::pub::DataWriter** (p. 1138) via **rti::core↩ ::policy::PublishMode::flow_controller_name** (p. 2016). A single **FlowController** (p. 1572) may service multiple DataWriters instances, even if they belong to a different **dds::pub::Publisher** (p. 1993). The `property` determines how the **FlowController** (p. 1572) shapes the network traffic.

Precondition

The specified `property` must be consistent, or the operation will fail and no **rti::pub::FlowController** (p. 1572) will be created.

See also

rti::pub::FlowControllerProperty (p. 1577) for rules on consistency among **property** (p. 1574)

8.130.3 Member Function Documentation

8.130.3.1 name()

```
std::string rti::pub::FlowController::name ( ) const  [inline]
```

Gets the name of this **FlowController** (p. 1572).

8.130.3.2 participant()

```
dds::domain::DomainParticipant rti::pub::FlowController::participant ( ) const  [inline]
```

Gets the participant associated to this **FlowController** (p. 1572).

8.130.3.3 property() [1/2]

```
FlowControllerProperty rti::pub::FlowController::property ( ) const  [inline]
```

Gets the configuration of this **FlowController** (p. 1572).

8.130.3.4 property() [2/2]

```
void rti::pub::FlowController::property (
            const  FlowControllerProperty & prop ) const  [inline]
```

Gets the configuration of this **FlowController** (p. 1572).

This operation modifies the property of the **rti::pub::FlowController** (p. 1572).

Once a **rti::pub::FlowController** (p. 1572) has been instantiated, only the **rti::pub::FlowControllerProperty::token_bucket** (p. 1579) can be changed. The **rti::pub::FlowControllerProperty::scheduling_policy** (p. 1579) is immutable.

A new **FlowControllerTokenBucketProperty::period** (p. 1587) only takes effect at the next scheduled token distribution time (as determined by its previous value).

Parameters

prop	<<***in***>> *(p. 355)* The new **rti::pub::FlowControllerProperty** (p. 1577). Property must be consistent. Immutable fields cannot be changed after **rti::pub::FlowController** (p. 1572) has been created.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::ImmutablePolicyError** (p. 1609), or **dds::core::InconsistentPolicyError** (p. 1610).

See also

rti::pub::FlowControllerProperty (p. 1577) for rules on consistency among **property** (p. 1574) values.

8.130.3.5 trigger_flow()

```
void rti::pub::FlowController::trigger_flow ( )  [inline]
```

Provides an external way to trigger a **FlowController** (p. 1572).

Typically, a **rti::pub::FlowController** (p. 1572) uses an internal trigger to periodically replenish its tokens. The period by which this trigger is called is determined by the **FlowControllerTokenBucketProperty::period** (p. 1587) property setting.

This function provides an additional, external trigger to the **rti::pub::FlowController** (p. 1572). This trigger adds **FlowControllerTokenBucketProperty::tokens_added_per_period** (p. 1585) tokens each time it is called (subject to the other property settings of the **rti::pub::FlowController** (p. 1572)).

An *on-demand* **rti::pub::FlowController** (p. 1572) can be created with a **dds::core::Duration::infinite()** (p. 1451) as **FlowControllerTokenBucketProperty::period** (p. 1587), in which case the only trigger source is external (i.e. the **FlowController** (p. 1572) is solely triggered by the user on demand).

rti::pub::FlowController::trigger_flow (p. 1575) can be called on both strict *on-demand* **FlowController** (p. 1572) and hybrid **FlowController** (p. 1572) (internally and externally triggered).

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.130.3.6 retain()

```
void rti::pub::FlowController::retain ( )  [inline]
```

Disables the automatic destruction of this object.

Disables the automatic destruction of the underlying **FlowController** (p. 1572) when when there are no more references to it. After that it can be looked up using **rti::pub::find_flow_controller** (p. 750)

To delete a retained object, manually call **close()** (p. 1576)

8.130.3.7 close()

```
void rti::pub::FlowController::close ( )  [inline]
```

Manually destroys this object.

Destroys the object referenced by this **FlowController** (p. 1572) reference.

After closing it, any calls to this object through this or other references throw **dds::core::AlreadyClosedError** (p. 806).

See also

retain() (p. 1576)

8.130.3.8 closed()

```
bool rti::pub::FlowController::closed ( ) const  [inline]
```

Returns true if this **FlowController** (p. 1572) has been closed.

The **FlowController** (p. 1572) may have been closed either by calling **close()** (p. 1576) or by closing the related DomainParticipant.

8.130.4 Friends And Related Function Documentation

8.130.4.1 find_flow_controller()

```
FlowController find_flow_controller (
            dds::domain::DomainParticipant participant,
           const std::string & name )  [related]
```

Retrieves an existing **FlowController** (p. 1572).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Parameters

participant	The DomainParticipant associated to the **FlowController** (p. 1572)
name	The name used to create the **FlowController** (p. 1572) or the name of one of the built-in FlowControllers (**FlowController::DEFAULT_NAME** (p. 58), **FlowController::FIXED_RATE_NAME** (p. 59), **FlowController::ON_DEMAND_NAME** (p. 60))

Returns

The flow controller with that name in that participant or an empty reference (equals to **dds::core::null** (p. 438)) if it doesn't exist

See also

FlowController::retain() (p. 1576)

8.131 rti::pub::FlowControllerProperty Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Configures a **FlowController** (p. 1572)

```
#include <FlowController.hpp>
```

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **FlowControllerProperty** ()

  *Creates a **FlowControllerProperty** (p. 1577) with earliest-deadline-first scheduling policy and default token-bucket configuration.*
- **FlowControllerProperty** (FlowControllerSchedulingPolicy::type **scheduling_policy**, const **FlowController↩ TokenBucketProperty** & **token_bucket**= **FlowControllerTokenBucketProperty**())

  *Creates a **FlowControllerProperty** (p. 1577) with the specified scheduling policy and token-bucket configuration.*
- FlowControllerSchedulingPolicy::type **scheduling_policy** () const

  Gets the scheduling policy.
- **FlowControllerProperty** & **scheduling_policy** (FlowControllerSchedulingPolicy::type value)

  Sets the scheduling policy.
- **FlowControllerTokenBucketProperty** & **token_bucket** ()

  Gets the token-bucket configuration by reference.
- const **FlowControllerTokenBucketProperty** & **token_bucket** () const

  Gets the token-bucket configuration by const-reference.

Static Public Member Functions

- static **FlowControllerProperty RoundRobin** ()

  *Creates a **FlowControllerProperty** (p. 1577) with round-robin scheduling policy and default token-bucket configuration.*
- static **FlowControllerProperty EarliestDeadlineFirst** ()

  *Creates a **FlowControllerProperty** (p. 1577) with earliest-deadline-first scheduling policy and default token-bucket configuration.*
- static **FlowControllerProperty HighestPriorityFirst** ()

  *Creates a **FlowControllerProperty** (p. 1577) with highest-priority-first scheduling policy and default token-bucket configuration.*

8.131.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Configures a **FlowController** (p. 1572)

The flow control characteristics shape the network traffic by determining how often and in what order associated asynchronous **dds::pub::DataWriter** (p. 1138) instances are serviced and how much data they are allowed to send.

Note that these settings apply directly to the **rti::pub::FlowController** (p. 1572), and do not depend on the number of **dds::pub::DataWriter** (p. 1138) instances the **rti::pub::FlowController** (p. 1572) is servicing. For instance, the specified flow rate does *not* double simply because two **dds::pub::DataWriter** (p. 1138) instances are waiting to write.

Entity:

rti::pub::FlowController (p. 1572)

Properties:

RxO (p. 514) = N/A

Changeable (p. 514) = **NO** (p. 514) for **rti::pub::FlowControllerProperty::scheduling_policy** (p. 1579), **YES** (p. 514) for **rti::pub::FlowControllerProperty::token_bucket** (p. 1579). However, the special value of **dds::core::Duration::infinite()** (p. 1451) as **FlowControllerTokenBucketProperty::period** (p. 1587) is strictly used to create an *on-demand* **rti::pub::FlowController** (p. 1572). The token period cannot toggle from an infinite to finite value (or vice versa). It can, however, change from one finite value to another.

8.131.2 Constructor & Destructor Documentation

8.131.2.1 FlowControllerProperty() [1/2]

```
rti::pub::FlowControllerProperty::FlowControllerProperty ( )  [inline]
```

Creates a **FlowControllerProperty** (p. 1577) with earliest-deadline-first scheduling policy and default token-bucket configuration.

8.131.2.2 FlowControllerProperty() [2/2]

```
rti::pub::FlowControllerProperty::FlowControllerProperty (
            FlowControllerSchedulingPolicy::type scheduling_policy,
            const  FlowControllerTokenBucketProperty & token_bucket =  FlowControllerToken↩
BucketProperty() )
```

Creates a **FlowControllerProperty** (p. 1577) with the specified scheduling policy and token-bucket configuration.

Parameters

scheduling_policy	The scheduling policy
token_bucket	The token-bucket configuration

8.131.3 Member Function Documentation

8.131.3.1 scheduling_policy() [1/2]

```
FlowControllerSchedulingPolicy::type rti::pub::FlowControllerProperty::scheduling_policy ( ) const
```

Gets the scheduling policy.

8.131.3.2 scheduling_policy() [2/2]

```
FlowControllerProperty & rti::pub::FlowControllerProperty::scheduling_policy (
            FlowControllerSchedulingPolicy::type value )
```

Sets the scheduling policy.

Returns

∗this

8.131.3.3 token_bucket() [1/2]

```
FlowControllerTokenBucketProperty & rti::pub::FlowControllerProperty::token_bucket ( )
```

Gets the token-bucket configuration by reference.

8.131.3.4 token_bucket() [2/2]

```
const FlowControllerTokenBucketProperty & rti::pub::FlowControllerProperty::token_bucket ( )
const
```

Gets the token-bucket configuration by const-reference.

8.131.3.5 RoundRobin()

```
static FlowControllerProperty rti::pub::FlowControllerProperty::RoundRobin ( ) [inline], [static]
```

Creates a **FlowControllerProperty** (p. 1577) with round-robin scheduling policy and default token-bucket configuration.

See also

FlowControllerSchedulingPolicy_def::ROUND_ROBIN (p. 1582)

8.131.3.6 EarliestDeadlineFirst()

```
static FlowControllerProperty rti::pub::FlowControllerProperty::EarliestDeadlineFirst ( ) [inline],
[static]
```

Creates a **FlowControllerProperty** (p. 1577) with earliest-deadline-first scheduling policy and default token-bucket configuration.

See also

FlowControllerSchedulingPolicy_def::EARLIEST_DEADLINE_FIRST (p. 1582)

8.131.3.7 HighestPriorityFirst()

```
static FlowControllerProperty rti::pub::FlowControllerProperty::HighestPriorityFirst ( ) [inline],
[static]
```

Creates a **FlowControllerProperty** (p. 1577) with highest-priority-first scheduling policy and default token-bucket configuration.

See also

FlowControllerSchedulingPolicy_def::HIGHEST_PRIORITY_FIRST (p. 1583)

8.132 rti::pub::FlowControllerSchedulingPolicy_def Struct Reference

<<***extension***>> *(*p. *354)* Kinds of flow controller shceduling policy

`#include <FlowController.hpp>`

Public Types

- enum **type** {
  ROUND_ROBIN ,
  EARLIEST_DEADLINE_FIRST ,
  HIGHEST_PRIORITY_FIRST }

  The underlying `enum` *type.*

8.132.1 Detailed Description

<<***extension***>> *(*p. *354)* Kinds of flow controller shceduling policy

Samples written by an asynchronous **dds::pub::DataWriter** (p. 1138) are not sent in the context of the **dds::pub::↩ DataWriter::write()** (p. 1148) call. Instead, the middleware puts the samples in a queue for future processing. The **rti↩ ::pub::FlowController** (p. 1572) associated with each asynchronous DataWriter instance determines when the samples are actually sent.

Each **rti::pub::FlowController** (p. 1572) maintains a separate FIFO queue for each unique destination (remote application). Samples written by asynchronous **dds::pub::DataWriter** (p. 1138) instances associated with the flow controller, are placed in the queues that correspond to the intended destinations of the sample.

When tokens become available, a flow controller must decide which queue(s) to grant tokens first. This is determined by the flow controller's scheduling policy. Once a queue has been granted tokens, it is serviced by the asynchronous publishing thread. The queued up samples will be coalesced and sent to the corresponding destination. The number of samples sent depends on the data size and the number of tokens granted.

QoS:

rti::pub::FlowControllerProperty (p. 1577)

See also

rti::pub::FlowControllerSchedulingPolicy (p. 58) The safe enumeration for this **type** (p. 1581)

8.132.2 Member Enumeration Documentation

8.132.2.1 type

`enum` **rti::pub::FlowControllerSchedulingPolicy_def::type**

The underlying `enum` type.

Enumerator

ROUND_ROBIN	Indicates to flow control in a round-robin fashion. Whenever tokens become available, the flow controller distributes the tokens uniformly across all of its (non-empty) destination queues. No destinations are prioritized. Instead, all destinations are treated equally and are serviced in a round-robin fashion.
EARLIEST_DEADLINE_FIRST	Indicates to flow control in an earliest-deadline-first fashion. A sample's deadline is determined by the time it was written plus the latency budget of the DataWriter at the time of the write call (as specified in the **dds::core::policy::LatencyBudget** (p. 1631)). The relative priority of a flow controller's destination queue is determined by the earliest deadline across all samples it contains. When tokens become available, the **rti::pub::FlowController** (p. 1572) distributes tokens to the destination queues in order of their deadline priority. In other words, the queue containing the sample with the earliest deadline is serviced first. The number of tokens granted equals the number of tokens required to send the first sample in the queue. Note that the priority of a queue may change as samples are sent (i.e. removed from the queue). If a sample must be sent to multiple destinations or two samples have an equal deadline value, the corresponding destination queues are serviced in a round-robin fashion. Hence, under the default **dds::core::policy::LatencyBudget::duration** (p. 1633) setting, an `EDF_FLOW_CONTROLLER_SCHED_POLICY` **rti::pub::FlowController** (p. 1572) preserves the order in which the user calls **dds::pub::DataWriter::write()** (p. 1148) across the DataWriters associated with the flow controller. Since the **dds::core::policy::LatencyBudget** (p. 1631) is mutable, a sample written second may contain an earlier deadline than the sample written first if the **dds::core::policy::LatencyBudget::duration** (p. 1633) value is sufficiently decreased in between writing the two samples. In that case, if the first sample is not yet written (still in queue waiting for its turn), it inherits the priority corresponding to the (earlier) deadline from the second sample. In other words, the priority of a destination queue is always determined by the earliest deadline among all samples contained in the queue. This priority inheritance approach is required in order to both honor the updated **dds::core::policy::LatencyBudget::duration** (p. 1633) and adhere to the **dds::pub::DataWriter** (p. 1138) in-order data delivery guarantee. **[default]** for **dds::pub::DataWriter** (p. 1138)

Enumerator

HIGHEST_PRIORITY_FIRST	Indicates to flow control in a highest-priority-first fashion. Determines the next destination queue to service as determined by the publication priority of the **dds::pub::DataWriter** (p. 1138), channel of multi-channel DataWriter, or individual sample. The relative priority of a flow controller's destination queue is determined by the highest publication priority of all samples it contains. When tokens become available, the **rti::pub::FlowController** (p. 1572) distributes tokens to the destination queues in order of their publication priority. In other words, the queue containing the sample with the highest publication priority is serviced first. The number of tokens granted equals the number of tokens required to send the first sample in the queue. Note that the priority of a queue may change as samples are sent (i.e. removed from the queue). If a sample must be sent to multiple destinations or two samples have an equal publication priority, the corresponding destination queues are serviced in a round-robin fashion. This priority inheritance approach is required in order to both honor the designated publication priority and adhere to the **dds::pub::DataWriter** (p. 1138) in-order data delivery guarantee.

8.133 rti::pub::FlowControllerTokenBucketProperty Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Configures a **FlowController** (p. 1572) based on a tocken-bucket mechanism

```
#include <FlowController.hpp>
```

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **FlowControllerTokenBucketProperty** (int32_t **max_tokens**= **dds::core::LENGTH_UNLIMITED**, int32_t **tokens_added_per_period**= **dds::core::LENGTH_UNLIMITED**, int32_t **tokens_leaked_per_period**=0, const **dds::core::Duration** & **period**= **dds::core::Duration**(1, 0), int32_t **bytes_per_token**= **dds::core::LENGTH_UNLIMITED**)

  Initializes the properties.
- int32_t **max_tokens** () const

  Gets the max_tokens.
- **FlowControllerTokenBucketProperty** & **max_tokens** (int32_t value)

  Sets the maximum number of tokens that can accumulate in the token bucket.
- int32_t **tokens_added_per_period** () const

  Gets the tokens_added_per_period.
- **FlowControllerTokenBucketProperty** & **tokens_added_per_period** (int32_t value)

  Sets the number of tokens added to the token bucket per specified period.
- int32_t **tokens_leaked_per_period** () const

  Gets the tokens_leaked_per_period.

- **FlowControllerTokenBucketProperty** & **tokens_leaked_per_period** (int32_t value)

  Sets the number of tokens removed from the token bucket per specified period.
- **dds::core::Duration period** () const

  Gets the period.
- **FlowControllerTokenBucketProperty** & **period** (const **dds::core::Duration** &value)

  Sets the period for adding tokens to and removing tokens from the bucket.
- int32_t **bytes_per_token** () const

  Gets the bytes_per_token.
- **FlowControllerTokenBucketProperty** & **bytes_per_token** (int32_t value)

  Sets the maximum number of bytes allowed to send for each token available.

8.133.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Configures a **FlowController** (p. 1572) based on a tocken-bucket mechanism

rti::pub::FlowController (p. 1572) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties.

Asynchronously published samples are queued up and transmitted based on the token bucket flow control scheme. The token bucket contains tokens, each of which represents a number of bytes. Samples can be sent only when there are sufficient tokens in the bucket. As samples are sent, tokens are consumed. The number of tokens consumed is proportional to the size of the data being sent. Tokens are replenished on a periodic basis.

The rate at which tokens become available and other token bucket properties determine the network traffic flow.

Note that if the same sample must be sent to multiple destinations, separate tokens are required for each destination. Only when multiple samples are destined to the same destination will they be co-alesced and sent using the same token(s). In other words, each token can only contribute to a single network packet.

Entity:

rti::pub::FlowController (p. 1572)

Properties:

RxO (p. 514) = N/A

Changeable (p. 514) = **YES** (p. 514). However, the special value of **dds::core::Duration::infinite()** (p. 1451) as **Flow↩ ControllerTokenBucketProperty::period** (p. 1587) is strictly used to create an *on-demand* **rti::pub::FlowController** (p. 1572). The token period cannot toggle from an infinite to finite value (or vice versa). It can, however, change from one finite value to another.

8.133.2 Constructor & Destructor Documentation

8.133.2.1 FlowControllerTokenBucketProperty()

```
rti::pub::FlowControllerTokenBucketProperty::FlowControllerTokenBucketProperty (
            int32_t max_tokens =  dds::core::LENGTH_UNLIMITED,
            int32_t tokens_added_per_period =  dds::core::LENGTH_UNLIMITED,
            int32_t tokens_leaked_per_period = 0,
            const  dds::core::Duration & period =  dds::core::Duration(1, 0),
            int32_t bytes_per_token =  dds::core::LENGTH_UNLIMITED )
```

Initializes the properties.

8.133.3 Member Function Documentation

8.133.3.1 max_tokens() [1/2]

```
int32_t rti::pub::FlowControllerTokenBucketProperty::max_tokens ( ) const
```

Gets the max_tokens.

See also

max_tokens(int32_t) (p. 1585)

8.133.3.2 max_tokens() [2/2]

```
FlowControllerTokenBucketProperty & rti::pub::FlowControllerTokenBucketProperty::max_tokens (
            int32_t value )
```

Sets the maximum number of tokens that can accumulate in the token bucket.

The number of tokens in the bucket will never exceed this value. Any excess tokens are discarded. This property value, combined with **FlowControllerTokenBucketProperty::bytes_per_token** (p. 1587), determines the maximum allowable data burst.

Use **dds::core::LENGTH_UNLIMITED** (p. 438) to allow accumulation of an unlimited amount of tokens (and therefore potentially an unlimited burst size).

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1,**dds::core::LENGTH_UNLIMITED** (p. 438)]

Returns

∗this

8.133.3.3 tokens_added_per_period() [1/2]

```
int32_t rti::pub::FlowControllerTokenBucketProperty::tokens_added_per_period ( ) const
```

Gets the tokens_added_per_period.

See also

tokens_added_per_period(int32_t) (p. 1586)

8.133.3.4 tokens_added_per_period() [2/2]

```
FlowControllerTokenBucketProperty & rti::pub::FlowControllerTokenBucketProperty::tokens_added_↩
per_period (
            int32_t value )
```

Sets the number of tokens added to the token bucket per specified period.

rti::pub::FlowController (p. 1572) transmits data only when tokens are available. Tokens are periodically replenished. This field determines the number of tokens added to the token bucket with each periodic replenishment.

Available tokens are distributed to associated **dds::pub::DataWriter** (p. 1138) instances based on the **rti::pub::Flow↩ ControllerProperty::scheduling_policy** (p. 1579).

Use **dds::core::LENGTH_UNLIMITED** (p. 438) to add the maximum number of tokens allowed by **FlowController↩ TokenBucketProperty::max_tokens** (p. 1585).

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1,**dds::core::LENGTH_UNLIMITED** (p. 438)]

Returns

∗this

8.133.3.5 tokens_leaked_per_period() [1/2]

```
int32_t rti::pub::FlowControllerTokenBucketProperty::tokens_leaked_per_period ( ) const
```

Gets the tokens_leaked_per_period.

See also

tokens_leaked_per_period(int32_t) (p. 1586)

8.133.3.6 tokens_leaked_per_period() [2/2]

```
FlowControllerTokenBucketProperty & rti::pub::FlowControllerTokenBucketProperty::tokens_leaked_↩
per_period (
            int32_t value )
```

Sets the number of tokens removed from the token bucket per specified period.

rti::pub::FlowController (p. 1572) transmits data only when tokens are available. When tokens are replenished and there are sufficient tokens to send all samples in the queue, this property determines whether any or all of the leftover tokens remain in the bucket.

Use **dds::core::LENGTH_UNLIMITED** (p. 438) to remove all excess tokens from the token bucket once all samples have been sent. In other words, no token accumulation is allowed. When new samples are written after tokens were purged, the earliest point in time at which they can be sent is at the next periodic replenishment.

[default] 0

[range] [0,**dds::core::LENGTH_UNLIMITED** (p. 438)]

Returns

∗this

8.133.3.7 period() [1/2]

```
dds::core::Duration rti::pub::FlowControllerTokenBucketProperty::period ( ) const
```

Gets the period.

See also

period(const dds::core::Duration&) (p. 1587)

8.133.3.8 period() [2/2]

```
FlowControllerTokenBucketProperty & rti::pub::FlowControllerTokenBucketProperty::period (
            const  dds::core::Duration & value )
```

Sets the period for adding tokens to and removing tokens from the bucket.

rti::pub::FlowController (p. 1572) transmits data only when tokens are available. This field determines the period by which tokens are added or removed from the token bucket.

The special value **dds::core::Duration::infinite()** (p. 1451) can be used to create an *on-demand* **rti::pub::Flow↩ Controller** (p. 1572), for which tokens are no longer replenished periodically. Instead, tokens must be added explicitly by calling **rti::pub::FlowController::trigger_flow** (p. 1575). This external trigger adds **FlowControllerTokenBucket↩ Property::tokens_added_per_period** (p. 1585) tokens each time it is called (subject to the other property settings).

[default] 1 second

[range] [1 nanosec, 1 year] or **dds::core::Duration::infinite()** (p. 1451)

Returns

∗this

8.133.3.9 bytes_per_token() [1/2]

```
int32_t rti::pub::FlowControllerTokenBucketProperty::bytes_per_token ( ) const
```

Gets the bytes_per_token.

See also

bytes_per_token(int32_t) (p. 1588)

8.133.3.10 bytes_per_token() [2/2]

```
FlowControllerTokenBucketProperty & rti::pub::FlowControllerTokenBucketProperty::bytes_per_token
(
            int32_t value )
```

Sets the maximum number of bytes allowed to send for each token available.

rti::pub::FlowController (p. 1572) transmits data only when tokens are available. This field determines the number of bytes that can actually be transmitted based on the number of tokens.

Tokens are always consumed in whole by each **dds::pub::DataWriter** (p. 1138). That is, in cases where **Flow↩ ControllerTokenBucketProperty::bytes_per_token** (p. 1587) is greater than the sample size, multiple samples may be sent to the same destination using a single token (regardless of **rti::pub::FlowControllerProperty::scheduling_↩ policy** (p. 1579)).

Where fragmentation is required, the fragment size will be **FlowControllerTokenBucketProperty::bytes_per_token** (p. 1587) or the minimum largest message size across all transports installed with the **dds::pub::DataWriter** (p. 1138), whichever is less.

Use **dds::core::LENGTH_UNLIMITED** (p. 438) to indicate that an unlimited number of bytes can be transmitted per token. In other words, a single token allows the recipient **dds::pub::DataWriter** (p. 1138) to transmit all its queued samples to a single destination. A separate token is required to send to each additional destination.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1024,**dds::core::LENGTH_UNLIMITED** (p. 438)]

Returns

∗this

8.134 Foo Class Reference

An example topic-type.

8.134.1 Detailed Description

An example topic-type.

This API documentation uses the type **Foo** (p. 1588) as a hypothetical **IDL-generated** (p. 606) **topic-type** (p. 444) for many code examples.

It is also used as a synonym of the template parameter of classes like DataWriter<T>.

Examples

Foo.hpp, and **Foo.idl**.

8.135 dds::sub::GenerationCount Class Reference

<<***value-type***>> *(*p. *350)*

#include <TGenerationCount.hpp>

Public Member Functions

- **GenerationCount** ()

  Create a default ***GenerationCount*** *(*p. *1589) object.*
- **GenerationCount** (int32_t disposed_count, int32_t no_writers_count)

  Create a ***GenerationCount*** *(*p. *1589) object with the provided disposed_count and no_writers count.*
- int32_t **disposed** () const

  Get the disposed generation count.
- int32_t **no_writers** () const

  Get the no_writers generation count.

8.135.1 Detailed Description

<<***value-type***>> *(*p. *350)*

8.135.2 Constructor & Destructor Documentation

8.135.2.1 GenerationCount() [1/2]

dds::sub::GenerationCount::GenerationCount ( ) [inline]

Create a default **GenerationCount** (p. 1589) object.

8.135.2.2 GenerationCount() [2/2]

```
dds::sub::GenerationCount::GenerationCount (
            int32_t disposed_count,
            int32_t no_writers_count )  [inline]
```

Create a **GenerationCount** (p. 1589) object with the provided disposed_count and no_writers count.

8.135.3 Member Function Documentation

8.135.3.1 disposed()

```
int32_t dds::sub::GenerationCount::disposed ( ) const  [inline]
```

Get the disposed generation count.

The disposed generation count of the instance at the time of sample reception.

Indicates how many times the `instance_state` of the corresponding instance changed from **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) to **dds::sub::status::InstanceState::alive()** (p. 1617). The counter is reset when the instance resource is reclaimed (removed from the **DataReader** (p. 976) cache).

See also

Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count (p. 2287)
Interpretation of the SampleInfo counters and ranks (p. 2288)

8.135.3.2 no_writers()

```
int32_t dds::sub::GenerationCount::no_writers ( ) const  [inline]
```

Get the no_writers generation count.

The no writers generation count of the instance at the time of sample reception.

Indicates how many times the `instance_state` of the corresponding instance changed from **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) to **dds::sub::status::InstanceState::alive()** (p. 1617). The counter is reset when the instance resource is reclaimed (removed from the **DataReader** (p. 976) cache).

See also

Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count (p. 2287)
Interpretation of the SampleInfo counters and ranks (p. 2288)

8.136 dds::core::policy::GroupData Class Reference

`#include <dds/core/policy/CorePolicy.hpp>`

Public Member Functions

- **GroupData** ()

  *Create a **GroupData** (p. 1591) instance.*
- **GroupData** (const **dds::core::ByteSeq** &seq)

  *Create a **GroupData** (p. 1591) instance with a sequence of bytes.*
- **GroupData** (const uint8_t *value_begin, const uint8_t *value_end)

  *Create a **GroupData** (p. 1591) instance with a sequence of bytes.*
- template<typename OCTET_ITER >
  GroupData & **value** (OCTET_ITER the_begin, OCTET_ITER the_end)

  *Sets the value for this **GroupData** (p. 1591).*
- const **dds::core::ByteSeq** **value** () const

  Getter (see setter with the same name)
- **dds::core::ByteSeq** & **value** (**dds::core::ByteSeq** &dst) const

  *Sets the value for this **GroupData** (p. 1591).*
- const uint8_t * **begin** () const

  Beginning of the range of bytes.
- const uint8_t * **end** () const

  End of the range of bytes.

8.136.1 Detailed Description

Entity:

dds::pub::Publisher (p. 1993), **dds::sub::Subscriber** (p. 2410)

Properties:

RxO (p. 514) = NO
Changeable (p. 514) = **YES** (p. 514)

See also

dds::sub::builtin_subscriber (p. 671)

8.136.2 Usage

The additional information is attached to a **dds::pub::Publisher** (p. 1993) or **dds::sub::Subscriber** (p. 2410). This extra data is not used by RTI Connext itself. When a remote application discovers the **dds::pub::Publisher** (p. 1993) or **dds::sub::Subscriber** (p. 2410), it can access that information and use it for its own purposes.

Use cases for this QoS policy, as well as the **dds::core::policy::TopicData** (p. 2500) and **dds::core::policy::UserData** (p. 2591), are often application-to-application identification, authentication, authorization, and encryption purposes. For example, applications can use Group or User Data to send security certificates to each other for RSA-type security.

In combination with **dds::sub::DataReaderListener** (p. 1055), **dds::pub::DataWriterListener** (p. 1202) and operations such as **dds::pub::ignore** (p. 647) and **dds::sub::ignore** (p. 667), this QoS policy can help an application to define and enforce its own security policies. For example, an application can implement matching policies similar to those of the **dds::core::policy::Partition** (p. 1923), except the decision can be made based on an application-defined policy.

The use of this QoS is not limited to security; it offers a simple, yet flexible extensibility mechanism.

Important: RTI Connext stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Connext with the maximum size of the data that will be stored in policies of this type. This size is configured with **rti::core::policy::DomainParticipantResourceLimits::publisher_group_data_max_length** (p. 1415) and **rti↩ ::core::policy::DomainParticipantResourceLimits::subscriber_group_data_max_length** (p. 1415).

See also

UserData (p. 2591)

8.136.3 Constructor & Destructor Documentation

8.136.3.1 GroupData() [1/3]

```
dds::core::policy::GroupData::GroupData ( )  [inline]
```

Create a **GroupData** (p. 1591) instance.

8.136.3.2 GroupData() [2/3]

```
dds::core::policy::GroupData::GroupData (
            const  dds::core::ByteSeq & seq )  [inline], [explicit]
```

Create a **GroupData** (p. 1591) instance with a sequence of bytes.

8.136.3.3 GroupData() [3/3]

```
dds::core::policy::GroupData::GroupData (
            const uint8_t * value_begin,
            const uint8_t * value_end )  [inline]
```

Create a **GroupData** (p. 1591) instance with a sequence of bytes.

8.136.4 Member Function Documentation

8.136.4.1 value() [1/3]

```
template<typename OCTET_ITER >
GroupData & dds::core::policy::GroupData::value (
            OCTET_ITER the_begin,
            OCTET_ITER the_end )  [inline]
```

Sets the value for this **GroupData** (p. 1591).

8.136.4.2 value() [2/3]

```
const  dds::core::ByteSeq dds::core::policy::GroupData::value ( ) const  [inline]
```

Getter (see setter with the same name)

8.136.4.3 value() [3/3]

```
 dds::core::ByteSeq & dds::core::policy::GroupData::value (
             dds::core::ByteSeq & dst ) const  [inline]
```

Sets the value for this **GroupData** (p. 1591).

8.136.4.4 begin()

```
const uint8_t * dds::core::policy::GroupData::begin ( ) const  [inline]
```

Beginning of the range of bytes.

8.136.4.5 end()

```
const uint8_t * dds::core::policy::GroupData::end ( ) const  [inline]
```

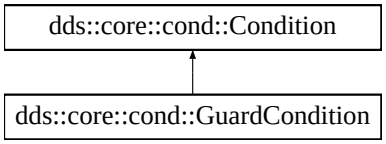
End of the range of bytes.

8.137 dds::core::cond::GuardCondition Class Reference

<<***reference-type***>> *(p. 351)* A condition whose trigger value is under the control of the application.

`#include <dds/core/cond/GuardCondition.hpp>`

Inheritance diagram for dds::core::cond::GuardCondition:



Public Member Functions

- template<typename Functor >
  void **handler** (const Functor &func)

  Registers a custom handler with this condition.
- void **reset_handler** ()

  Resets the handler for this condition.
- void **trigger_value** (bool value)

  Manually sets the trigger value.

8.137.1 Detailed Description

<<***reference-type***>> *(p. 351)* A condition whose trigger value is under the control of the application.

The **dds::core::cond::GuardCondition** (p. 1594) provides a way for an application to manually wake up a **dds::core::cond::WaitSet** (p. 2617). This is accomplished by attaching the **dds::core::cond::GuardCondition** (p. 1594) to the **dds::core::cond::WaitSet** (p. 2617) and then setting the `trigger_value` by means of the **dds::core::cond::GuardCondition::trigger_value()** (p. 1595) operation.

See also

dds::core::cond::WaitSet (p. 2617)

8.137.2 Member Function Documentation

8.137.2.1 handler()

```
template<typename Functor >
void dds::core::cond::GuardCondition::handler (
             const Functor & func )  [inline]
```

Registers a custom handler with this condition.

For information about condition handlers, see **this ReadCondition constructor** (p. 2134)

Parameters

func	The handler to be registered with this condition. This argument is copied.

MT Safety:

It is not safe to call **handler()** (p. 1594) or **reset_handler()** (p. 1595) in parallel. It is not safe to call **handler()** (p. 1594) or **reset_handler()** (p. 1595) at the same time as **trigger_value()** (p. 1595). It is not safe to call **handler()** (p. 1594) or **reset_handler()** (p. 1595) while this condition is attached to a **WaitSet** (p. 2617) that is being dispatched.

8.137.2.2 reset_handler()

```
void dds::core::cond::GuardCondition::reset_handler ( )  [inline]
```

Resets the handler for this condition.

After the invocation of this method no handler will be registered with this condition.

MT Safety:

It is not safe to call **handler()** (p. 1594), **reset_handler()** (p. 1595) or **trigger_value()** (p. 1595) concurrently.

8.137.2.3 trigger_value()

```
void dds::core::cond::GuardCondition::trigger_value (
             bool value )  [inline]
```

Manually sets the trigger value.

Setting the trigger value to true causes the activation of the condition in a **WaitSet** (p. 2617).

MT Safety:

Safe in all situations except when **trigger_value()** (p. 1595) is called in parallel to **handler()** (p. 1594) or **reset_↩ handler()** (p. 1595).

See also

WaitSet::wait() (p. 2622)

WaitSet::dispatch() (p. 2624)

8.138 rti::core::Guid Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Class for GUID (Global Unique Identifier) representation

`#include "rti/core/Guid.hpp"`

Public Member Functions

- **Guid** ()
  *Create a default **Guid** (p. 1596) representing the unknown guid.*
- uint8_t & **operator[ ]** (uint32_t index)
  *Get a reference to value of the **Guid** (p. 1596) at position index.*
- const uint8_t & **operator[ ]** (uint32_t index) const
  *Get const reference to the value of the **Guid** (p. 1596) at position index.*
- bool **operator**< (const **Guid** &other) const
  Compare two Guids for a less-than relationship.
- bool **operator**<**=** (const **Guid** &other) const
  Compare two Guids for a less-than-or-equal relationship.
- bool **operator**> (const **Guid** &other) const
  Compare two Guids for a greater-than relationship.
- bool **operator**>**=** (const **Guid** &other) const
  Compare two Guids for a greater-than-or-equal relationship.

Static Public Member Functions

- static **Guid unknown** ()
  Unknown GUID.
- static **Guid zero** ()
  Zero GUID.
- static **Guid automatic** ()
  *Construct a **Guid** (p. 1596) indicating that RTI Connext should choose an appropriate virtual GUID.*

Static Public Attributes

- static const size_t **LENGTH** = 16
  *The number of elements of a **Guid** (p. 1596).*

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator**<< (std::ostream &out, const **Guid** &guid)
  *Prints a **Guid** (p. 1596) to an output stream.*

8.138.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Class for GUID (Global Unique Identifier) representation

8.138.2 Constructor & Destructor Documentation

8.138.2.1 Guid()

```
rti::core::Guid::Guid ( )  [inline]
```

Create a default **Guid** (p. 1596) representing the unknown guid.

8.138.3 Member Function Documentation

8.138.3.1 unknown()

```
static  Guid rti::core::Guid::unknown ( )  [inline], [static]
```

Unknown GUID.

8.138.3.2 zero()

```
static  Guid rti::core::Guid::zero ( )  [inline], [static]
```

Zero GUID.

8.138.3.3 automatic()

```
static  Guid rti::core::Guid::automatic ( )  [inline], [static]
```

Construct a **Guid** (p. 1596) indicating that RTI Connext should choose an appropriate virtual GUID.

Indicates that RTI Connext should choose an appropriate virtual GUID.

If this special value is assigned to **rti::core::policy::DataWriterProtocol::virtual_guid** (p. 1212) or **rti::core::policy↩ ::DataReaderProtocol::virtual_guid** (p. 1060), RTI Connext will assign the virtual GUID automatically based on the RTPS or physical GUID.

8.138.3.4 operator[]() [1/2]

```
uint8_t & rti::core::Guid::operator[] (
            uint32_t index )
```

Get a reference to value of the **Guid** (p. 1596) at position index.

8.138.3.5 operator[]() [2/2]

```
const uint8_t & rti::core::Guid::operator[] (
            uint32_t index ) const
```

Get const reference to the value of the **Guid** (p. 1596) at position index.

8.138.3.6 operator<()

```
bool rti::core::Guid::operator< (
            const  Guid & other ) const
```

Compare two Guids for a less-than relationship.

8.138.3.7 operator<=()

```
bool rti::core::Guid::operator<= (
            const  Guid & other ) const
```

Compare two Guids for a less-than-or-equal relationship.

8.138.3.8 operator>()

```
bool rti::core::Guid::operator> (
            const  Guid & other ) const
```

Compare two Guids for a greater-than relationship.

8.138.3.9 operator>=()

```
bool rti::core::Guid::operator>= (
            const  Guid & other ) const
```

Compare two Guids for a greater-than-or-equal relationship.

8.138.4 Friends And Related Function Documentation

8.138.4.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const  Guid & guid )  [related]
```

Prints a **Guid** (p. 1596) to an output stream.

8.138.5 Member Data Documentation

8.138.5.1 LENGTH

```
const size_t rti::core::Guid::LENGTH = 16  [static]
```

The number of elements of a **Guid** (p. 1596).

8.139 rti::util::heap_monitoring::HeapMonitoringParams Class Reference

Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.

```
#include <util.hpp>
```

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **HeapMonitoringParams** (**SnapshotOutputFormat snapshot_output_format**=rti::util::heap_monitoring::↩ SnapshotOutputFormat::STANDARD, **SnapshotContentFormat snapshot_content_format**=rti::util::heap_↩ monitoring::SnapshotContentFormat::DEFAULT)

  *Create **HeapMonitoringParams** (p. 1599).*
- **SnapshotOutputFormat snapshot_output_format** () const

  Get the format of the output of the snapshot.
- void **snapshot_output_format** (**SnapshotOutputFormat** snapshot_output_format)

  Set the format of the output of the snapshot.
- **SnapshotContentFormat snapshot_content_format** () const

  Get the bitmap to decide which information of the snapshot will be displayed.
- void **snapshot_content_format** (**SnapshotContentFormat** snapshot_content_format)

  Set the bitmap to decide which information of the snapshot will be displayed.

8.139.1 Detailed Description

Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.

<<***extension***>> *(p. 354)* Input parameters for enabling heap monitoring They will be used for configuring the format of the snapshot.

8.139.2 Constructor & Destructor Documentation

8.139.2.1 HeapMonitoringParams()

```
rti::util::heap_monitoring::HeapMonitoringParams::HeapMonitoringParams (
            SnapshotOutputFormat snapshot_output_format = rti::util::heap_monitoring::SnapshotOutputFormat↩
::STANDARD,
            SnapshotContentFormat snapshot_content_format = rti::util::heap_monitoring::SnapshotContentFormat↩
::DEFAULT )
```

Create **HeapMonitoringParams** (p. 1599).

Parameters

snapshot_output_format	The format of the output of the snapshot.
snapshot_content_format	The bitmap to decide which information of the snapshot will be displayed.

8.139.3 Member Function Documentation

8.139.3.1 snapshot_output_format() [1/2]

```
SnapshotOutputFormat rti::util::heap_monitoring::HeapMonitoringParams::snapshot_output_format ( )
const
```

Get the format of the output of the snapshot.

Returns

The format of the output of the snapshot

8.139.3.2 snapshot_output_format() [2/2]

```
void rti::util::heap_monitoring::HeapMonitoringParams::snapshot_output_format (
            SnapshotOutputFormat snapshot_output_format )
```

Set the format of the output of the snapshot.

Parameters

snapshot_output_format	The format of the output of the snapshot

8.139.3.3 snapshot_content_format() [1/2]

```
SnapshotContentFormat rti::util::heap_monitoring::HeapMonitoringParams::snapshot_content_format (
) const
```

Get the bitmap to decide which information of the snapshot will be displayed.

Returns

The bitmap to decide which information of the snapshot will be displayed.

8.139.3.4 snapshot_content_format() [2/2]

```
void rti::util::heap_monitoring::HeapMonitoringParams::snapshot_content_format (
            SnapshotContentFormat snapshot_content_format )
```

Set the bitmap to decide which information of the snapshot will be displayed.

Parameters

snapshot_content_format	The bitmap to decide which information of the snapshot will be displayed.

8.140 dds::core::policy::History Class Reference

Specifies how much historical data a **dds::pub::DataWriter** (p. 1138) and a **dds::sub::DataReader** (p. 976) can store.

`#include <dds/core/policy/CorePolicy.hpp>`

Public Member Functions

- **History** ()
  Creates a policy that keeps the last sample only.
- **History** (**dds::core::policy::HistoryKind** the_kind, int32_t the_depth=1)
  Creates a policy with a specific history kind and optionally a history depth.
- **History** & **kind** (**dds::core::policy::HistoryKind** the_kind)
  Sets the history kind.
- **dds::core::policy::HistoryKind kind** () const
  Gets the history kind.
- int32_t **depth** () const
  Gets the history depth.
- **History** & **depth** (int32_t the_depth)
  Sets the history depth.

Static Public Member Functions

- static **History KeepAll** ()
  *Creates a **History** (p. 1602) with HistoryKind::KEEP_ALL.*
- static **History KeepLast** (uint32_t **depth**)
  *Creates a **History** (p. 1602) with HistoryKind::KEEP_LAST and the specified depth.*

8.140.1 Detailed Description

Specifies how much historical data a **dds::pub::DataWriter** (p. 1138) and a **dds::sub::DataReader** (p. 976) can store.

This QoS policy specifies how much data must to stored by RTI Connext for a **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976). It controls whether RTI Connext should deliver only the most recent value, attempt to deliver all intermediate values, or do something in between.

On the publishing side, this QoS policy controls the samples that should be maintained by the **dds::pub::DataWriter** (p. 1138) on behalf of existing **dds::sub::DataReader** (p. 976) entities. The behavior with regards to a **dds::sub::DataReader** (p. 976) entities discovered after a sample is written is controlled by the **DURABILITY** (p. 529) policy.

On the subscribing side, this QoS policy controls the samples that should be maintained until the application "takes" them from RTI Connext.

Entity:

dds::topic::Topic (p. 2474), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Properties:

RxO (p. 514) = NO
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

See also

dds::core::policy::Reliability (p. 2149)

dds::core::policy::History (p. 1602)

8.140.2 Usage

This policy controls the behavior of RTI Connext when the value of an instance changes before it is finally communicated to **dds::sub::DataReader** (p. 976) entities.

When a **dds::pub::DataWriter** (p. 1138) sends data, or a **dds::sub::DataReader** (p. 976) receives data, the data sent or received is stored in a cache whose contents are controlled by this QoS policy. This QoS policy interacts with **dds::core::policy::Reliability** (p. 2149) by controlling whether RTI Connext guarantees that *all* of the sent data is received (**dds::core::policy::HistoryKind_def::KEEP_ALL** (p. 1607)) or if only the last N data values sent are guaranteed to be received (**dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607))–this is a reduced level of reliability.

The amount of data that is sent to new DataReaders who have configured their **dds::core::policy::Durability** (p. 1433) to receive previously published data is controlled by **dds::core::policy::Durability::writer_depth** (p. 1439), not by the **History** (p. 1602) QoS policy.

Note that the **History** (p. 1602) QoS policy does not control the *physical* sizes of the send and receive queues. The memory allocation for the queues is controlled by the **dds::core::policy::ResourceLimits** (p. 2203).

If `kind` is **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607) (the default), then RTI Connext will only attempt to keep the latest values of the instance and discard the older ones. In this case, the value of `depth` regulates the maximum number of values (up to and including the most current one) RTI Connext will maintain and deliver until the samples are fully acknowledged. After N values have been sent or received, any new data will overwrite the oldest data in the queue. Thus the queue acts like a circular buffer of length *N*.

For keyed-data, there is different behavior on the publishing and subscribing sides associated with how invalid samples representing the disposal of or unregistration from an instance affect history.

On the publishing side, unregistering from or disposing of an instance creates an invalid sample that is accounted for in the history depth. This means that an invalid sample may replace a value that is currently being stored in the writer queue.

On the subscribing side, however, invalid samples do not count towards history depth and will not replace a value that is being stored in the reader queue.

On both the publishing and subscribing sides, there can only ever be one invalid sample per-instance and that one sample can be in different states depending on whether the instance has been disposed, unregistered, or both.

The default (and most common setting) for `depth` is 1, indicating that only the most recent value should be delivered.

If `kind` is **dds::core::policy::HistoryKind_def::KEEP_ALL** (p. 1607), then RTI Connext will attempt to maintain and deliver all the values of the instance to existing subscribers. The resources that RTI Connext can use to keep this history are limited by the settings of the **RESOURCE_LIMITS** (p. 546). If the limit is reached, then the behavior of RTI Connext will depend on the **RELIABILITY** (p. 544). If the **Reliability** (p. 2149) `kind` is **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157), then the old values will be discarded. If **Reliability** (p. 2149) `kind` is RELIABLE, then RTI Connext will block the **dds::pub::DataWriter** (p. 1138) until it can deliver the necessary old values to all subscribers.

8.140.3 Consistency

This QoS policy's `depth` must be consistent with the **RESOURCE_LIMITS** (p. 546) `max_samples_per_↩ instance`. For these two QoS to be consistent, they must verify that *depth <= max_samples_per_instance*.

See also

dds::core::policy::ResourceLimits (p. 2203)

8.140.4 Constructor & Destructor Documentation

8.140.4.1 History() [1/2]

```
dds::core::policy::History::History ( )  [inline]
```

Creates a policy that keeps the last sample only.

8.140.4.2 History() [2/2]

```
dds::core::policy::History::History (
             dds::core::policy::HistoryKind the_kind,
            int32_t the_depth = 1 )  [inline]
```

Creates a policy with a specific history kind and optionally a history depth.

The history depth doesn't apply to HistoryKind::KEEP_ALL

8.140.5 Member Function Documentation

8.140.5.1 kind() [1/2]

```
History & dds::core::policy::History::kind (
             dds::core::policy::HistoryKind the_kind )  [inline]
```

Sets the history kind.

*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* Specifies the kind of history to be kept.

For DataWriters, the samples are only kept either until they are fully acknowledged by all matching DataReaders or until they are replaced or removed. Samples can be replaced or removed for a number of reasons, including but not limited to **dds::core::policy::Lifespan** (p. 1635) expiration, replacement because the **dds::core::policy::History↩ ::depth** (p. 1605) has been exceeded, or one of the **dds::core::policy::ResourceLimits** (p. 2203) settings has been exceeded.

[default] dds::core::policy::HistoryKind_def::KEEP_LAST (p. 1607)

Categories

Immutable (p. 243), **Applied Per-Instance** (p. 287), **Reliability** (p. 222), **Data Availability** (p. 167)

8.140.5.2 kind() [2/2]

```
dds::core::policy::HistoryKind dds::core::policy::History::kind ( ) const  [inline]
```

Gets the history kind.

8.140.5.3 depth() [1/2]

```
int32_t dds::core::policy::History::depth ( ) const  [inline]
```

Gets the history depth.

8.140.5.4 depth() [2/2]

```
History & dds::core::policy::History::depth (
            int32_t the_depth )  [inline]
```

Sets the history depth.

<<***basic***>> *(*p. *104)* <<***spec***>> *(*p. *112)* Specifies the number of samples per instance to be kept, when the `kind` is **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607)

If a value other than 1 (the default) is specified, it should be consistent with the settings of the **RESOURCE_LIMITS** (p. 546) policy. That is:

depth <= **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207)

When the `kind` is **dds::core::policy::HistoryKind_def::KEEP_ALL** (p. 1607), the depth has no effect. Its implied value is `infinity` (in practice limited by the settings of the **RESOURCE_LIMITS** (p. 546) policy).

When a DataWriter responds to a TopicQuery, the samples that are evaluated against the TopicQuery filter are only those samples that fall within the **dds::core::policy::Durability::writer_depth** (p. 1439), not this depth.

[default] 1

[range] [1,100 million], <= **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207)

Categories

Immutable (p. 243), **Applied Per-Instance** (p. 287), **Reliability** (p. 222), **Data Availability** (p. 167)

8.140.5.5 KeepAll()

```
static History dds::core::policy::History::KeepAll ( ) [inline], [static]
```

Creates a **History** (p. 1602) with HistoryKind::KEEP_ALL.

8.140.5.6 KeepLast()

```
static History dds::core::policy::History::KeepLast (
            uint32_t depth ) [inline], [static]
```

Creates a **History** (p. 1602) with HistoryKind::KEEP_LAST and the specified depth.

8.141 dds::core::policy::HistoryKind_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) HistoryKind.

`#include <PolicyKind.hpp>`

Public Types

- enum **type** {
  KEEP_LAST ,
  KEEP_ALL }
  The underlying `enum` type.

8.141.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) HistoryKind.

8.141.2 Member Enumeration Documentation

8.141.2.1 type

`enum` **dds::core::policy::HistoryKind_def::type**

The underlying `enum` type.

Enumerator

KEEP_LAST	**[default]** Keep the last `depth` samples. On the publishing side, RTI Connext will only attempt to keep the most recent `depth` samples of each instance of data (identified by its key) managed by the **dds::pub::DataWriter** (p. 1138). Invalid samples representing a disposal or unregistration of an instance count towards the depth and may replace other DDS samples currently in the DataWriter queue for the same instance. On the subscribing side, the **dds::sub::DataReader** (p. 976) will only attempt to keep the most recent `depth` samples received for each instance (identified by its key) until the application takes them via the **dds::sub::DataReader** (p. 976) 's **take() (**p. **1024)** operation. Invalid samples representing a disposal or unregistration of an instance do not count towards the history depth and will not replace other DDS samples currently in the DataReader queue for the same instance.
KEEP_ALL	Keep *all* the samples. On the publishing side, RTI Connext will attempt to keep all samples (representing each value written) of each instance of data (identified by its key) managed by the **dds::pub::DataWriter** (p. 1138) until they can be delivered to all subscribers. On the subscribing side, RTI Connext will attempt to keep all samples of each instance of data (identified by its key) managed by the **dds::sub::DataReader** (p. 976). These samples are kept until the application takes them from RTI Connext via the **take() (**p. **1024)** operation.

8.142 rti::core::policy::IgnoredEntityReplacementKind_def Struct Reference

<<***extension***>> *(*p. *354)* The enumeration for **DomainParticipantResourceLimits** (p. 1391) ignored entity replacement kinds

`#include <PolicyKind.hpp>`

Public Types

- enum **type**

  The underlying `enum` type.

8.142.1 Detailed Description

<<***extension***>> *(*p. *354)* The enumeration for **DomainParticipantResourceLimits** (p. 1391) ignored entity replacement kinds

8.142.2 Member Enumeration Documentation

8.142.2.1 type

`enum` **rti::core::policy::IgnoredEntityReplacementKind_def::type**

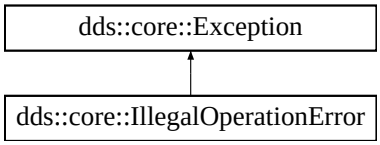
The underlying `enum` type.

8.143 dds::core::IllegalOperationError Class Reference

Indicates that an operation was called under improper circumstances.

`#include <Exception.hpp>`

Inheritance diagram for dds::core::IllegalOperationError:



Public Member Functions

- virtual const char * **what** () const throw ()

  *Access the message contained in this **IllegalOperationError** (p. 1608) exception.*

8.143.1 Detailed Description

Indicates that an operation was called under improper circumstances.

Inherits also from `std::logic_error`

An operation was invoked on an inappropriate object or at an inappropriate time. This return code is similar to **dds::core::PreconditionNotMetError** (p. 1941), except that there is no precondition that could be changed to make the operation succeed.

8.143.2 Member Function Documentation

8.143.2.1 what()

```
virtual const char * dds::core::IllegalOperationError::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **IllegalOperationError** (p. 1608) exception.

Returns

The message.

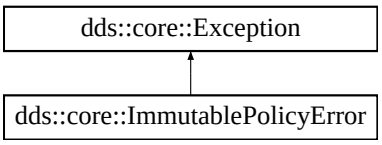
Implements **dds::core::Exception** (p. 1547).

8.144 dds::core::ImmutablePolicyError Class Reference

Indicates that the application attempted to modify an immutable QoS policy.

```
#include <Exception.hpp>
```

Inheritance diagram for dds::core::ImmutablePolicyError:



Public Member Functions

- virtual const char * **what** () const throw ()
  *Access the message contained in this **ImmutablePolicyError** (p. 1609) exception.*

8.144.1 Detailed Description

Indicates that the application attempted to modify an immutable QoS policy.

Inherits also from `std::logic_error`

8.144.2 Member Function Documentation

8.144.2.1 what()

```
virtual const char * dds::core::ImmutablePolicyError::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **ImmutablePolicyError** (p. 1609) exception.

Returns

The message.

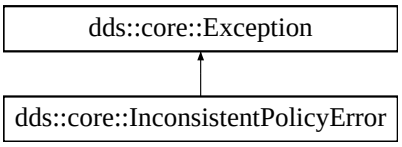
Implements **dds::core::Exception** (p. 1547).

8.145 dds::core::InconsistentPolicyError Class Reference

Indicates that the application specified a set of QoS policies that are not consistent with each other.

```
#include <Exception.hpp>
```

Inheritance diagram for dds::core::InconsistentPolicyError:



Public Member Functions

- virtual const char * **what** () const throw ()
  *Access the message contained in this **NotEnabledError** (p. 1869) exception.*

8.145.1 Detailed Description

Indicates that the application specified a set of QoS policies that are not consistent with each other.

Inherits also from `std::logic_error`

8.145.2 Member Function Documentation

8.145.2.1 what()

```
virtual const char * dds::core::InconsistentPolicyError::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **NotEnabledError** (p. 1869) exception.

Returns

The message.

Implements **dds::core::Exception** (p. 1547).

8.146 dds::core::status::InconsistentTopicStatus Class Reference

Information about the status **dds::core::status::StatusMask::inconsistent_topic()** (p. 2379)

```
#include <TStatus.hpp>
```

Public Member Functions

- int32_t **total_count** () const

  *Total cumulative count of the pairs (**dds::sub::DataReader** (p. 976),**dds::pub::DataWriter** (p. 1138)) whose topic names match the **dds::topic::Topic** (p. 2474) to which this status is attached and whose types are inconsistent according to the rules defined in **dds::core::policy::TypeConsistencyEnforcement** (p. 2563).*
- int32_t **total_count_change** () const

  *The incremental number of inconsistent pairs (**dds::sub::DataReader** (p. 976),**dds::pub::DataWriter** (p. 1138)) for the **dds::topic::Topic** (p. 2474) to which this status is attached, that have been discovered since the last time this status was read.*

8.146.1 Detailed Description

Information about the status **dds::core::status::StatusMask::inconsistent_topic()** (p. 2379)

Entity:

dds::topic::Topic (p. 2474)

Listener:

TopicListener

Every time a **dds::sub::DataReader** (p. 976) and **dds::pub::DataWriter** (p. 1138) with the same **dds::topic::Topic** (p. 2474) do not match because the type-consistency enforcement policy fails, the inconsistent topic status is updated for the **dds::topic::Topic** (p. 2474).

See also

dds::core::policy::TypeConsistencyEnforcement (p. 2563)

8.146.2 Member Function Documentation

8.146.2.1 total_count()

```
int32_t dds::core::status::InconsistentTopicStatus::total_count ( ) const  [inline]
```

Total cumulative count of the pairs (**dds::sub::DataReader** (p. 976),**dds::pub::DataWriter** (p. 1138)) whose topic names match the **dds::topic::Topic** (p. 2474) to which this status is attached and whose types are inconsistent according to the rules defined in **dds::core::policy::TypeConsistencyEnforcement** (p. 2563).

8.146.2.2 total_count_change()

```
int32_t dds::core::status::InconsistentTopicStatus::total_count_change ( ) const  [inline]
```

The incremental number of inconsistent pairs (**dds::sub::DataReader** (p. 976),**dds::pub::DataWriter** (p. 1138)) for the **dds::topic::Topic** (p. 2474) to which this status is attached, that have been discovered since the last time this status was read.

8.147 dds::core::InstanceHandle Class Reference

<<***value-type***>> *(p. 350)* Handle to identify different instances of the same **dds::topic::Topic** (p. 2474) of a certain type.

```
#include <dds/core/InstanceHandle.hpp>
```

Public Member Functions

- **InstanceHandle** (const **dds::core::null_type** &null_handle)

  Create a nil instance handle.
- bool **operator==** (const **dds::core::null_type** &) const

  *Returns **is_nil()** (p. 1614)*
- bool **is_nil** () const

  *Check if this **InstanceHandle** (p. 1612) represents the nil **InstanceHandle** (p. 1612).*

Static Public Member Functions

- static **InstanceHandle** **nil** ()

  Returns the nil instance handle.

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator**<< (std::ostream &os, const **dds::core::InstanceHandle** &h)
  Prints an instance handle.

8.147.1 Detailed Description

<<***value-type***>> *(*p. *350)* Handle to identify different instances of the same **dds::topic::Topic** (p. 2474) of a certain type.

Handle to identify different instances of the same **dds::topic::Topic** (p. 2474) of a certain type.

See also

dds::pub::DataWriter::register_instance (p. 1158)
dds::sub::SampleInfo::instance_handle (p. 2290)

Note

A specialization of std::hash for **InstanceHandle** (p. 1612) is defined.

See also

dds::pub::DataWriter::register_instance (p. 1158)
dds::sub::SampleInfo::instance_handle (p. 2290)

8.147.2 Constructor & Destructor Documentation

8.147.2.1 InstanceHandle()

```
dds::core::InstanceHandle::InstanceHandle (
            const  dds::core::null_type & null_handle )  [inline]
```

Create a nil instance handle.

8.147.3 Member Function Documentation

8.147.3.1 operator==()

```
bool dds::core::InstanceHandle::operator== (
            const  dds::core::null_type &  ) const  [inline]
```

Returns **is_nil()** (p. 1614)

8.147.3.2 nil()

```
static  InstanceHandle dds::core::InstanceHandle::nil ( )  [inline], [static]
```

Returns the nil instance handle.

Special **dds::core::InstanceHandle** (p. 1612) value

See also

dds::core::InstanceHandle::is_nil (p. 1614)

8.147.3.3 is_nil()

```
bool dds::core::InstanceHandle::is_nil ( ) const  [inline]
```

Check if this **InstanceHandle** (p. 1612) represents the nil **InstanceHandle** (p. 1612).

Returns

true if the **InstanceHandle** (p. 1612) is nil, false otherwise

8.147.4 Friends And Related Function Documentation

8.147.4.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const  dds::core::InstanceHandle & h )  [related]
```

Prints an instance handle.

The format is 16 2-digit contiguous hexadecimal numbers, for example:
0000004d000000000000000000000000

Examples

Foo.hpp.

8.148 dds::sub::status::InstanceState Class Reference

Indicates if the samples are from a live **dds::pub::DataWriter** (p. 1138) or not.

`#include <dds/sub/status/DataState.hpp>`

Inherits std::bitset< OMG_DDS_STATE_BIT_COUNT >.

Public Types

- typedef std::bitset< OMG_DDS_STATE_BIT_COUNT > **MaskType**
  An std::bitset of InstanceStates.

Public Member Functions

- **InstanceState** (const **MaskType** &other)
  *Create an **InstanceState** (p. 1615) from MaskType.*

Static Public Member Functions

- static const **InstanceState** **alive** ()
  *Creates an alive **InstanceState** (p. 1615) object.*
- static const **InstanceState** **not_alive_disposed** ()
  *Creates a not_alive_disposed **InstanceState** (p. 1615) object.*
- static const **InstanceState** **not_alive_no_writers** ()
  *Creates a not_alive_no_writers **InstanceState** (p. 1615) object.*
- static const **InstanceState** **not_alive_mask** ()
  *Creates an **InstanceState** (p. 1615) object representing any not alive state.*
- static const **InstanceState** **any** ()
  *Creates an **InstanceState** (p. 1615) object representing any instance state.*

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator<<** (std::ostream &os, const **InstanceState** &s)
  Prints an instance state as a readable string.

8.148.1 Detailed Description

Indicates if the samples are from a live **dds::pub::DataWriter** (p. 1138) or not.

For each instance, the middleware internally maintains an instance state. The instance state can be:

- **dds::sub::status::InstanceState::alive()** (p. 1617) indicates that (a) samples have been received for the instance, (b) there are live **dds::pub::DataWriter** (p. 1138) entities writing the instance, and (c) the instance has not been explicitly disposed (or else more samples have been received after it was disposed).

- **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) indicates the instance was explicitly disposed by a **dds::pub::DataWriter** (p. 1138) by means of the dispose operation.

- **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) indicates the instance has been declared as not-alive by the **dds::sub::DataReader** (p. 976) because it detected that there are no live **dds::pub::DataWriter** (p. 1138) entities writing that instance.

The precise behavior events that cause the instance state to change depends on the setting of the OWNERSHIP QoS:

- If **OWNERSHIP** (p. 538) is set to **dds::core::policy::OwnershipKind_def::EXCLUSIVE** (p. 1906), then the instance state becomes **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) only if the **dds::pub::DataWriter** (p. 1138) that "owns" the instance explicitly disposes it. The instance state becomes **dds::sub::status::InstanceState::alive()** (p. 1617) again only if the **dds::pub::DataWriter** (p. 1138) that owns the instance writes it.

- If **OWNERSHIP** (p. 538) is set to **dds::core::policy::OwnershipKind_def::SHARED** (p. 1906), then the instance state becomes **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) if any **dds::pub::DataWriter** (p. 1138) explicitly disposes the instance. The instance state becomes **dds::sub::status::InstanceState::alive()** (p. 1617) as soon as any **dds::pub::DataWriter** (p. 1138) writes the instance again.

The instance state available in the **dds::sub::SampleInfo** (p. 2285) is a snapshot of the instance state of the instance at the time the collection was obtained (i.e. at the time read or take was called). The instance state is therefore the same for all samples in the returned collection that refer to the same instance.

For example, you can check if an instance is disposed as follows:

```
for (auto sample : reader.take()) {
    if (sample.info().state().instance_state() ==
            dds::sub::status::InstanceState::not_alive_disposed()) {
        // ...
    }
}
```

8.148.2 Member Typedef Documentation

8.148.2.1 MaskType

```
typedef std::bitset<OMG_DDS_STATE_BIT_COUNT >  dds::sub::status::InstanceState::MaskType
```

An std::bitset of InstanceStates.

8.148.3 Constructor & Destructor Documentation

8.148.3.1 InstanceState()

```
dds::sub::status::InstanceState::InstanceState (
            const  MaskType & other )  [inline]
```

Create an **InstanceState** (p. 1615) from MaskType.

Parameters

other	The MaskType to create the **InstanceState** (p. 1615) with

8.148.4 Member Function Documentation

8.148.4.1 alive()

```
static const  InstanceState dds::sub::status::InstanceState::alive ( )  [inline], [static]
```

Creates an alive **InstanceState** (p. 1615) object.

The instance is currently in existence.

Referenced by **dds::sub::status::DataState::any_data()**, **dds::sub::status::DataState::new_data()**, and **dds::sub::status::DataState::new_instance()**.

8.148.4.2 not_alive_disposed()

```
static const  InstanceState dds::sub::status::InstanceState::not_alive_disposed ( )  [inline],
[static]
```

Creates a not_alive_disposed **InstanceState** (p. 1615) object.

Indicates that the instance has been disposed by a DataWriter.

8.148.4.3 not_alive_no_writers()

```
static const  InstanceState dds::sub::status::InstanceState::not_alive_no_writers ( )  [inline],
[static]
```

Creates a not_alive_no_writers **InstanceState** (p. 1615) object.

Indicates that none of the DataWriters that are currently alive (according to the **dds::core::policy::Liveliness** (p. 1645) policy) are writing the instance.

8.148.4.4 not_alive_mask()

```
static const  InstanceState dds::sub::status::InstanceState::not_alive_mask ( )  [inline], [static]
```

Creates an **InstanceState** (p. 1615) object representing any not alive state.

This is equivalent to **not_alive_disposed()** (p. 1617) | **not_alive_no_writers()** (p. 1617)

8.148.4.5 any()

```
static const  InstanceState dds::sub::status::InstanceState::any ( )  [inline], [static]
```

Creates an **InstanceState** (p. 1615) object representing any instance state.

This is equivalent to **not_alive_mask()** (p. 1618) | **alive()** (p. 1617).

Referenced by **dds::sub::status::DataState::any()**.

8.148.5 Friends And Related Function Documentation

8.148.5.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const  InstanceState & s )  [related]
```

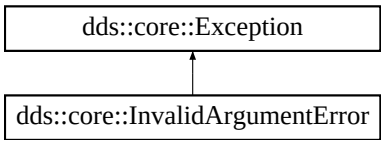
Prints an instance state as a readable string.

8.149 dds::core::InvalidArgumentError Class Reference

Indicates that the application passed an illegal parameter value into an operation.

`#include <Exception.hpp>`

Inheritance diagram for dds::core::InvalidArgumentError:



Public Member Functions

- virtual const char * **what** () const throw ()

  *Access the message contained in this **InvalidArgumentError** (p. 1619) exception.*

8.149.1 Detailed Description

Indicates that the application passed an illegal parameter value into an operation.

Inherits also from `std::invalid_argument`

8.149.2 Member Function Documentation

8.149.2.1 what()

```
virtual const char * dds::core::InvalidArgumentError::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **InvalidArgumentError** (p. 1619) exception.

Returns

The message.

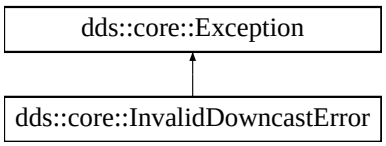
Implements **dds::core::Exception** (p. 1547).

8.150 dds::core::InvalidDowncastError Class Reference

Indicates that a downcast was incorrect.

`#include <Exception.hpp>`

Inheritance diagram for dds::core::InvalidDowncastError:



Public Member Functions

- virtual const char * **what** () const throw ()

  *Access the message contained in this **InvalidDowncastError** (p. 1620) exception.*

8.150.1 Detailed Description

Indicates that a downcast was incorrect.

Inherits also from `std::runtime_error`

See also

dds::core::polymorphic_cast (p. 620)

dds::pub::AnyDataWriter::get() (p. 819)

8.150.2 Member Function Documentation

8.150.2.1 what()

```
virtual const char * dds::core::InvalidDowncastError::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **InvalidDowncastError** (p. 1620) exception.

Returns

The message.

Implements **dds::core::Exception** (p. 1547).

8.151 rti::core::status::InvalidLocalAdvanceNoticeStatus Class Reference

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::invalid_local_advance_↩ notice()** (p. 2388)

`#include <Status.hpp>`

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **dds::core::Time expiration_time** () const
  The time when the local artifact will expire.
- unsigned int **expiration_code** () const
  The reason (a numeric code) for the local artifact to expire.
- const char * **expiration_message** () const
  *A descriptive message for the **rti::core::status::InvalidLocalAdvanceNoticeStatus::expiration_code** (p. 1621) code.*

8.151.1 Detailed Description

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::invalid_local_advance_↩ notice()** (p. 2388)

Currently, this status is only triggered when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

8.151.2 Member Function Documentation

8.151.2.1 expiration_time()

```
dds::core::Time rti::core::status::InvalidLocalAdvanceNoticeStatus::expiration_time ( ) const
[inline]
```

The time when the local artifact will expire.

8.151.2.2 expiration_code()

```
unsigned int rti::core::status::InvalidLocalAdvanceNoticeStatus::expiration_code ( ) const  [inline]
```

The reason (a numeric code) for the local artifact to expire.

8.151.2.3 expiration_message()

```
const char * rti::core::status::InvalidLocalAdvanceNoticeStatus::expiration_message ( ) const
[inline]
```

A descriptive message for the **rti::core::status::InvalidLocalAdvanceNoticeStatus::expiration_code** (p. 1621) code.

8.152 dds::topic::is_topic_type< T > Struct Template Reference

Trait that indicates if a type is suitable to be the type of a **dds::topic::Topic** (p. 2474).

```
#include <TopicTraits.hpp>
```

Inherits dds::core::false_type.

8.152.1 Detailed Description

template<typename T>
struct dds::topic::is_topic_type< T >

Trait that indicates if a type is suitable to be the type of a **dds::topic::Topic** (p. 2474).

When a type T meets this requirement the is_topic_type<T> inherits from `dds::core::true_type` and defines a member constant `is_topic_type<T>::value` that is true. For all other types it inherits from `dds::core::false_type` and defines `value` to false.

Types **generated by rtiddsgen** (p. 606) (except for those annotated with `@nested`), **dds::core::xtypes::DynamicData** (p. 1461) and the **Builtin types** (p. 50) all meet this requirement.

Attempting to instantiate a dds::topic::Topic<Foo> if is_topic_type<Foo>::value is false results in a compilation error.

8.153 rti::request::IsReplyRelatedPredicate< T > Class Template Reference

Functor to match replies by their related request.

```
#include <util.hpp>
```

Public Member Functions

- **IsReplyRelatedPredicate** (const **rti::core::SampleIdentity** &related_request_id)
  Creates the predicate with the request id to match.
- bool **operator()** (**dds::sub::Sample**< T > sample)
  Determines if a reply is related to the request id this object contains.
- bool **operator()** (**rti::sub::LoanedSample**< T > sample)
  Determines if a reply is related to the request id this object contains.
- bool **operator()** (**dds::sub::SampleInfo** sample_info)
  Determines if a reply info is related to the request id this object contains.

8.153.1 Detailed Description

template<typename T>
class rti::request::IsReplyRelatedPredicate< T >

Functor to match replies by their related request.

Useful for C++ standard algorithms like `std::find_if()`

See also

Correlating requests and replies (p. 346)

8.153.2 Constructor & Destructor Documentation

8.153.2.1 IsReplyRelatedPredicate()

```
template<typename T >
rti::request::IsReplyRelatedPredicate< T >::IsReplyRelatedPredicate (
            const  rti::core::SampleIdentity & related_request_id )  [inline]
```

Creates the predicate with the request id to match.

8.153.3 Member Function Documentation

8.153.3.1 operator()() [1/3]

```
template<typename T >
bool  rti::request::IsReplyRelatedPredicate< T >::operator() (
             dds::sub::Sample< T > sample )  [inline]
```

Determines if a reply is related to the request id this object contains.

References **dds::sub::Sample< T >::info()**, and **dds::sub::SampleInfo::related_original_publication_virtual_↩ sample_identity()**.

8.153.3.2 operator()() [2/3]

```
template<typename T >
bool  rti::request::IsReplyRelatedPredicate< T >::operator() (
              rti::sub::LoanedSample< T > sample )  [inline]
```

Determines if a reply is related to the request id this object contains.

References **rti::sub::LoanedSample< T >::info()**, and **dds::sub::SampleInfo::related_original_publication_virtual_sample_identity()**.

8.153.3.3 operator()() [3/3]

```
template<typename T >
bool  rti::request::IsReplyRelatedPredicate< T >::operator() (
              dds::sub::SampleInfo sample_info )  [inline]
```

Determines if a reply info is related to the request id this object contains.

References **dds::sub::SampleInfo::related_original_publication_virtual_sample_identity()**.

8.154 rti::sub::IsValidData< T > Struct Template Reference

<<***extension***>> *(*p. *354)* A functor that returns true when a sample has valid data.

```
#include <LoanedSamplesImpl.hpp>
```

Public Types

- typedef LoanedSamples< T >::value_type **sample_type**

  LoanedSample *(*p. *1659) type.*

Public Member Functions

- bool **operator()** (const **sample_type** &sample)

  Returns true if this sample contains valid data.

8.154.1 Detailed Description

template<typename T>
struct rti::sub::IsValidData< T >

<<***extension***>> *(p. 354)* A functor that returns true when a sample has valid data.

This functor is useful in algorithms that iterate on LoanedSamples. For example:

```
LoanedSamples<Foo> samples = reader.take();
std::cout « "Valid sample count: "
         « std::count_if(samples.begin(), samples.end(), IsValidData<Foo>());
```

A similar utility is the iterator **valid_data()** (p. 766).

See also

dds::sub::LoanedSamples (p. 1663)

valid_data() (p. 766)

8.154.2 Member Typedef Documentation

8.154.2.1 sample_type

```
template<typename T >
typedef LoanedSamples<T>::value_type  rti::sub::IsValidData< T >::sample_type
```

LoanedSample (p. 1659) type.

8.154.3 Member Function Documentation

8.154.3.1 operator()()

```
template<typename T >
bool  rti::sub::IsValidData< T >::operator() (
            const  sample_type & sample )  [inline]
```

Returns true if this sample contains valid data.

That is, if `sample.info()`.valid().

8.155 dds::core::KeyedBytesTopicType Class Reference

Built-in type consisting of a variable-length array of opaque bytes and a string that is the key.

`#include <dds/core/BuiltinTopicTypes.hpp>`

Public Member Functions

- **KeyedBytesTopicType** ()
  Creates a sample with an empty array of bytes and an empty string.
- **KeyedBytesTopicType** (const std::string &the_key, const std::vector< uint8_t > &the_value)
  Creates a sample with the given key and bytes.
- const **dds::core::string** & **key** () const
  Gets the key.
- **dds::core::string** & **key** ()
  Gets the key.
- void **key** (const **dds::core::string** &the_value)
  Sets the key.
- **operator std::vector< uint8_t >** () const
  Conversion to std::vector by obtaining the bytes.
- std::vector< uint8_t > **value** () const
  Gets the bytes.
- void **value** (const std::vector< uint8_t > &the_value)
  Sets the bytes.
- uint8_t & **operator[ ]** (uint32_t index)
  Accesses the bytes by index.
- uint8_t **operator[ ]** (uint32_t index) const
  Accesses the bytes by index.
- int32_t **length** () const
  Get the number of bytes.

8.155.1 Detailed Description

Built-in type consisting of a variable-length array of opaque bytes and a string that is the key.

8.155.2 Constructor & Destructor Documentation

8.155.2.1 KeyedBytesTopicType() [1/2]

`dds::core::KeyedBytesTopicType::KeyedBytesTopicType ( ) [inline]`

Creates a sample with an empty array of bytes and an empty string.

8.155.2.2 KeyedBytesTopicType() [2/2]

```
dds::core::KeyedBytesTopicType::KeyedBytesTopicType (
            const std::string & the_key,
            const std::vector< uint8_t > & the_value )  [inline]
```

Creates a sample with the given key and bytes.

8.155.3 Member Function Documentation

8.155.3.1 key() [1/3]

```
const  dds::core::string & dds::core::KeyedBytesTopicType::key ( ) const  [inline]
```

Gets the key.

8.155.3.2 key() [2/3]

```
 dds::core::string & dds::core::KeyedBytesTopicType::key ( )  [inline]
```

Gets the key.

8.155.3.3 key() [3/3]

```
void dds::core::KeyedBytesTopicType::key (
            const  dds::core::string & the_value )  [inline]
```

Sets the key.

References **dds::core::Value< D >::delegate()**.

8.155.3.4 operator std::vector< uint8_t >()

```
dds::core::KeyedBytesTopicType::operator std::vector< uint8_t > ( ) const  [inline]
```

Conversion to std::vector by obtaining the bytes.

References **dds::core::Value< D >::delegate()**.

8.155.3.5 value() [1/2]

```
std::vector< uint8_t > dds::core::KeyedBytesTopicType::value ( ) const  [inline]
```

Gets the bytes.

References **dds::core::Value< D >::delegate()**.

8.155.3.6 value() [2/2]

```
void dds::core::KeyedBytesTopicType::value (
            const std::vector< uint8_t > & the_value )  [inline]
```

Sets the bytes.

References **dds::core::Value< D >::delegate()**.

8.155.3.7 operator[]() [1/2]

```
uint8_t & dds::core::KeyedBytesTopicType::operator[] (
            uint32_t index )  [inline]
```

Accesses the bytes by index.

References **dds::core::Value< D >::delegate()**.

8.155.3.8 operator[]() [2/2]

```
uint8_t dds::core::KeyedBytesTopicType::operator[] (
            uint32_t index ) const  [inline]
```

Accesses the bytes by index.

References **dds::core::Value< D >::delegate()**.

8.155.3.9 length()

```
int32_t dds::core::KeyedBytesTopicType::length ( ) const  [inline]
```

Get the number of bytes.

References **dds::core::Value< D >::delegate()**.

8.156 dds::core::KeyedStringTopicType Class Reference

Built-in type consisting of a string payload and a second string that is the key.

`#include <dds/core/BuiltinTopicTypes.hpp>`

Public Member Functions

- **KeyedStringTopicType** ()
  Creates a sample with two empty strings.
- **KeyedStringTopicType** (const **dds::core::string** &the_key, const **dds::core::string** &the_value)
  Creates a sample with the two given strings.
- const **dds::core::string** & **key** () const
  Gets the key.
- **dds::core::string** & **key** ()
  Gets the key.
- void **key** (const **dds::core::string** &the_value)
  Sets the key.
- const **dds::core::string** & **value** () const
  Gets the value.
- **dds::core::string** & **value** ()
  Gets the key.
- void **value** (const **dds::core::string** &the_value)
  Gets the key.

8.156.1 Detailed Description

Built-in type consisting of a string payload and a second string that is the key.

See also

Built-in Types (p. 50)

8.156.2 Constructor & Destructor Documentation

8.156.2.1 KeyedStringTopicType() [1/2]

`dds::core::KeyedStringTopicType::KeyedStringTopicType ( ) [inline]`

Creates a sample with two empty strings.

8.156.2.2 KeyedStringTopicType() [2/2]

```
dds::core::KeyedStringTopicType::KeyedStringTopicType (
            const  dds::core::string & the_key,
            const  dds::core::string & the_value )  [inline]
```

Creates a sample with the two given strings.

8.156.3 Member Function Documentation

8.156.3.1 key() [1/3]

```
const  dds::core::string & dds::core::KeyedStringTopicType::key ( ) const  [inline]
```

Gets the key.

8.156.3.2 key() [2/3]

```
 dds::core::string & dds::core::KeyedStringTopicType::key ( )  [inline]
```

Gets the key.

References **dds::core::Value< D >::delegate()**.

8.156.3.3 key() [3/3]

```
void dds::core::KeyedStringTopicType::key (
            const  dds::core::string & the_value )  [inline]
```

Sets the key.

References **dds::core::Value< D >::delegate()**.

8.156.3.4 value() [1/3]

```
const  dds::core::string & dds::core::KeyedStringTopicType::value ( ) const  [inline]
```

Gets the value.

References **dds::core::Value< D >::delegate()**.

8.156.3.5 value() [2/3]

```
 dds::core::string & dds::core::KeyedStringTopicType::value ( )  [inline]
```

Gets the key.

References **dds::core::Value< D >::delegate()**.

8.156.3.6 value() [3/3]

```
void dds::core::KeyedStringTopicType::value (
             const  dds::core::string & the_value )  [inline]
```

Gets the key.

References **dds::core::Value< D >::delegate()**.

8.157 dds::core::policy::LatencyBudget Class Reference

Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **LatencyBudget** ()
  Creates a latency budget with zero duration.
- **LatencyBudget** (const **dds::core::Duration** &the_duration)
  Creates an instance with the specified duration.
- **LatencyBudget** & **duration** (const **dds::core::Duration** &the_duration)
  Sets the duration of the maximum acceptable delay.
- const **dds::core::Duration duration** () const
  Getter (see setter with the same name)

8.157.1 Detailed Description

Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

This policy is a *hint* to a DDS implementation; it can be used to change how it processes and sends data that has low latency requirements. The DDS specification does not mandate whether or how this policy is used.

Entity:

dds::topic::Topic (p. 2474), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Status:

dds::core::status::StatusMask::offered_incompatible_qos() (p. 2380), **dds::core::status::StatusMask↩ ::requested_incompatible_qos()** (p. 2381)

Properties:

RxO (p. 514) = YES
Changeable (p. 514) = **YES** (p. 514)

See also

rti::core::policy::PublishMode (p. 2013)

rti::pub::FlowController (p. 1572)

8.157.2 Usage

This policy provides a means for the application to indicate to the middleware the urgency of the data communication. By having a non-zero `duration`, RTI Connext can optimize its internal operation.

RTI Connext uses it in conjunction with **rti::core::policy::PublishModeKind_def::ASYNCHRONOUS** (p. 2019) **dds::pub::DataWriter** (p. 1138) instances associated with a FlowControllerSchedulingPolicy_def::EARLIEST_↩ DEADLINE_FIRST **rti::pub::FlowController** (p. 1572) only. Together with the time of write, **dds::core::policy↩ ::LatencyBudget::duration** (p. 1633) determines the deadline of each individual sample. RTI Connext uses this information to prioritize the sending of asynchronously published data; see **rti::core::policy::AsynchronousPublisher** (p. 832).

8.157.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered duration* <= *requested duration* evaluates to 'TRUE'.

8.157.4 Constructor & Destructor Documentation

8.157.4.1 LatencyBudget() [1/2]

```
dds::core::policy::LatencyBudget::LatencyBudget ( )  [inline]
```

Creates a latency budget with zero duration.

8.157.4.2 LatencyBudget() [2/2]

```
dds::core::policy::LatencyBudget::LatencyBudget (
            const  dds::core::Duration & the_duration )  [inline], [explicit]
```

Creates an instance with the specified duration.

8.157.5 Member Function Documentation

8.157.5.1 duration() [1/2]

```
LatencyBudget & dds::core::policy::LatencyBudget::duration (
            const  dds::core::Duration & the_duration )  [inline]
```

Sets the duration of the maximum acceptable delay.

[default] 0 (meaning minimize the delay)

Categories

Mutable (p. 280), **Discovery Metadata** (p. 288), **Matching** (p. 293), **Data Availability** (p. 167)

8.157.5.2 duration() [2/2]

```
const  dds::core::Duration dds::core::policy::LatencyBudget::duration ( ) const  [inline]
```

Getter (see setter with the same name)

8.158 rti::config::LibraryVersion Class Reference

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* The version of a single library shipped as part of an RTI Connext distribution.

`#include <rti/config/Version.hpp>`

Public Member Functions

- int32_t **major_version** () const

  Get the major version of a single RTI Connext library.
- int32_t **minor_version** () const

  Get the minor version of a single RTI Connext library.
- char **release_version** () const

  Get the release letter of a single RTI Connext library.
- int32_t **build_version** () const

  Get the build number of a single RTI Connext library.

8.158.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* The version of a single library shipped as part of an RTI Connext distribution.

RTI Connext is comprised of a number of separate libraries. Although RTI Connext as a whole has a version, the individual libraries each have their own versions as well. It may be necessary to check these individual library versions when seeking technical support.

8.158.2 Member Function Documentation

8.158.2.1 major_version()

`int32_t rti::config::LibraryVersion::major_version ( ) const`

Get the major version of a single RTI Connext library.

8.158.2.2 minor_version()

`int32_t rti::config::LibraryVersion::minor_version ( ) const`

Get the minor version of a single RTI Connext library.

8.158.2.3 release_version()

```
char rti::config::LibraryVersion::release_version ( ) const
```

Get the release letter of a single RTI Connext library.

8.158.2.4 build_version()

```
int32_t rti::config::LibraryVersion::build_version ( ) const
```

Get the build number of a single RTI Connext library.

8.159 dds::core::policy::Lifespan Class Reference

Specifies how long the data written by a **dds::pub::DataWriter** (p. 1138) is considered valid.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **Lifespan** ()
  Creates the default policy with an infinite lifespan.
- **Lifespan** (const **dds::core::Duration** &d)
  Creates a policy with the specified lifespan duration.
- **Lifespan** & **duration** (const **dds::core::Duration** &d)
  Sets the maximum duration for which the data is valid.
- const **dds::core::Duration** **duration** () const
  Getter (see setter with the same name)

8.159.1 Detailed Description

Specifies how long the data written by a **dds::pub::DataWriter** (p. 1138) is considered valid.

Each data sample written by the **dds::pub::DataWriter** (p. 1138) has an associated expiration time beyond which the data should not be delivered to any application. Once the sample expires, the data will be removed from the **dds::sub↩ ::DataReader** (p. 976) caches as well as from the transient and persistent information caches.

The expiration time of each sample from the **dds::pub::DataWriter** (p. 1138)'s cache is computed by adding the duration specified by this QoS policy to the time when the sample is added to the **dds::pub::DataWriter** (p. 1138)'s cache. This timestamp is not necessarily equal to the sample's source timestamp that can be provided by the user using the **dds::pub::DataWriter::write(const T&,const dds::core::Time&)** (p. 1149) or **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179) API.

The expiration time of each sample from the **dds::sub::DataReader** (p. 976)'s cache is computed by adding the duration to the reception timestamp.

See also

dds::pub::DataWriter::write() (p. 1148)

dds::pub::DataWriter::write(const T&,const dds::core::Time&) (p. 1149)

Entity:

dds::topic::Topic (p. 2474), **dds::pub::DataWriter** (p. 1138)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **YES** (p. 514)

8.159.2 Usage

The **Lifespan** (p. 1635) QoS policy can be used to control how much data is stored by RTI Connext. Even if it is configured to store "all" of the data sent or received for a topic (see **dds::core::policy::History** (p. 1602)), the total amount of data it stores may be limited by this QoS policy.

You may also use this QoS policy to ensure that applications do not receive or act on data, commands or messages that are too old and have 'expired.'

To avoid inconsistencies, multiple writers of the same instance should have the same lifespan.

See also

dds::sub::SampleInfo::source_timestamp (p. 2289)

dds::sub::SampleInfo::reception_timestamp (p. 2290)

8.159.3 Constructor & Destructor Documentation

8.159.3.1 Lifespan() [1/2]

```
dds::core::policy::Lifespan::Lifespan ( )  [inline]
```

Creates the default policy with an infinite lifespan.

8.159.3.2 Lifespan() [2/2]

```
dds::core::policy::Lifespan::Lifespan (
            const  dds::core::Duration & d )  [inline], [explicit]
```

Creates a policy with the specified lifespan duration.

8.159.4 Member Function Documentation

8.159.4.1 duration() [1/2]

```
Lifespan & dds::core::policy::Lifespan::duration (
            const  dds::core::Duration & d )  [inline]
```

Sets the maximum duration for which the data is valid.

[default] dds::core::Duration::infinite() (p. 1451)

[range] [1 nanosec, 1 year] or **dds::core::Duration::infinite()** (p. 1451)

Categories

Mutable (p. 280), **Discovery Metadata** (p. 288), **Durability** (p. 184), **Data Availability** (p. 167)

8.159.4.2 duration() [2/2]

```
const  dds::core::Duration dds::core::policy::Lifespan::duration ( ) const  [inline]
```

Getter (see setter with the same name)

8.160 Listener Class Reference

Entity listeners.

```
#include <dds/dds.hpp>
```

8.160.1 Detailed Description

Entity listeners.

Note

This type doesn't exist in the API, it just represents the concept that all subclasses of **dds::core::Entity** (p. 1520) have an associated listener type. The actual listener types are:

- **dds::domain::DomainParticipantListener** (p. 1382)
- **dds::topic::TopicListener** (p. 2508)
- **dds::sub::SubscriberListener** (p. 2422)
- **dds::pub::PublisherListener** (p. 2006)
- **dds::sub::DataReaderListener** (p. 1055)
- **dds::pub::DataWriterListener** (p. 1202)

Listeners provide a way for RTI Connext to asynchronously alert the application when there are relevant status changes.

Each dedicated listener presents a list of operations that correspond to the relevant communication status changes to which an application may respond.

The same **Listener** (p. 1637) instance may be shared among multiple entities if you so desire. Consequently, the provided parameter contains a reference to the concerned **dds::core::Entity** (p. 1520).

8.160.2 Access to Plain Communication Status

The general mapping between the plain communication statuses (see **Status Kinds** (p. 429)) and the listeners' operations is as follows:

- For each communication status, there is a corresponding operation whose name is `on_<communication_status>()`, which takes a parameter of type `<communication_status>` as listed in **Status Kinds** (p. 429).

- `on_<communication_status>` is available on the relevant **dds::core::Entity** (p. 1520) as well as those that embed it, as expressed in the following figure:

listener processing. The most *specific* relevant enabled listener is called."

- When the application attaches a listener on an entity, it must set a mask. The mask indicates to RTI Connext which operations are enabled within the listener (cf. operation **dds::core::Entity** (p. 1520) set_listener()).

- When a plain communication status changes, RTI Connext triggers the most specific relevant listener operation that is enabled. In case the most specific relevant listener operation corresponds to an application-installed 'nil' listener the operation will be considered handled by a NO-OP operation that does not reset the communication status.

This behavior allows the application to set a default behavior (e.g., in the listener associated with the **dds::domain::DomainParticipant** (p. 1326)) and to set dedicated behaviors only where needed.

8.160.3 Access to Read Communication Status

The two statuses related to data arrival are treated slightly differently. Since they constitute the core purpose of the Data Distribution Service, there is no need to provide a default mechanism (as is done for the plain communication statuses above).

The rule is as follows. Each time the read communication status changes:

- First, RTI Connext tries to trigger the dds::sub::SubscriberListener::on_data_on_readers with a parameter of the related **dds::sub::Subscriber** (p. 2410);

- If this does not succeed (there is no listener or the operation is not enabled), RTI Connext tries to trigger **dds::sub::DataReaderListener::on_data_available** (p. 1058) on all the related **dds::sub::DataReaderListener** (p. 1055) objects, with a parameter of the related **dds::sub::DataReader** (p. 976).

The rationale is that either the application is interested in relations among data arrivals and it must use the first option (and then get the corresponding **dds::sub::DataReader** (p. 976) objects by calling **dds::sub::find** (p. 672) on the related **dds::sub::Subscriber** (p. 2410) and then get the data by calling **dds::sub::DataReader::read** (p. 996) or **dds::sub::DataReader::take** (p. 997) on the returned **dds::sub::DataReader** (p. 976) objects), or it wants to treat each **dds::sub::DataReader** (p. 976) independently and it may choose the second option (and then get the data by calling **dds::sub::DataReader::read** (p. 996) or **dds::sub::DataReader::take** (p. 997) on the related **dds::sub::DataReader** (p. 976)).

Note that if dds::sub::SubscriberListener::on_data_on_readers is called, RTI Connext will *not* try to call **dds::sub::DataReaderListener::on_data_available** (p. 1058). However, an application can force a call to the **dds::sub::DataReader** (p. 976) objects that have data by calling **dds::sub::Subscriber::notify_datareaders** (p. 2414).

8.160.4 How to set a listener

Listeners can be set in the Entity constructor, or the `set_listener()` function (for example, **dds::sub::DataReader::set_listener()** (p. 1006)).

These functions expect a `std::shared_ptr`. The Entity keeps a reference to the `std::shared_ptr` so the listener cannot be deleted until the Entity is destroyed or the listener is reset with `set_listener(nullptr)`.

Warning

It is not recommended to keep a reference to an Entity inside of its listener because it creates a cycle that prevents the destruction of both objects. If you need to keep a reference, make sure you call `entity.set_listener(nullptr)` or `entity.**close()** (p. 1024)` explicitly.

8.160.5 Operations Allowed in Listener Callbacks

See `Restricted Operations in Listener Callbacks`, in the Core Libraries User's Manual.

8.160.6 Best Practices with Listeners

Note that all the issues described below are avoided by using **dds::core::cond::WaitSet** (p. 2617).

Avoid blocking or performing a lot of processing in Listener (p. **1637) callbacks**

Listeners are invoked by internal threads that perform critical functions within the middleware and need to run in a timely manner. By default, Connext DDS creates a few threads to use to receive data and only a single thread to handle periodic events.

Because of this, user applications installing Listeners should never block in a **Listener** (p. 1637) callback. There are several negative consequences of blocking in a listener callback:

- The application may lose data for the DataReader the listener is installed on, because the receive thread is not removing it from the socket buffer and it gets overwritten.
- The application may receive strictly reliable data with a delay, because the receive thread is not removing it from the socket buffer and if it gets overwritten it must be re-sent.
- The application may lose or delay data for other DataReaders, because by default all DataReaders created with the same DomainParticipant share the same threads.
- The application may not be notified of periodic events on time

If the application needs to make a blocking call when data is available, or when another event occurs, the application should use **dds::core::cond::WaitSet** (p. 2617).

Avoid taking application mutexes/semaphores in Listener (p. **1637) callbacks**

Taking application mutexes/sempahores within a **Listener** (p. 1637) callback may lead to unexpected deadlock scenarios.

When a **Listener** (p. 1637) callback is invoked the EA (Exclusive Area) of the Entity 'E' to which the callback applies is taken by the middleware.

If the application takes an application mutex 'M' within a critical section in which the application makes DDS calls affecting 'E', this may lead to following deadlock:

The middleware thread is within the entity EA trying to acquire the mutex 'M'. At the same time, the application thread has acquired 'M' and is blocked trying to acquire the entity EA.

Do not write data with a DataWriter within the on_data_available callback

Avoid writing data with a DataWriter within the **dds::sub::DataReaderListener::on_data_available()** (p. 1058) callback. If the write operation blocks because e.g. the send window is full, this will lead to a deadlock.

Do not call wait_for_acknowledgements within the on_data_available callback

Do not call the **dds::pub::DataWriter::wait_for_acknowledgments** (p. 1168) within the **dds::sub::DataReader↩ Listener::on_data_available()** (p. 1058) callback. This will lead to deadlock.

See also

Status Kinds (p. 429)

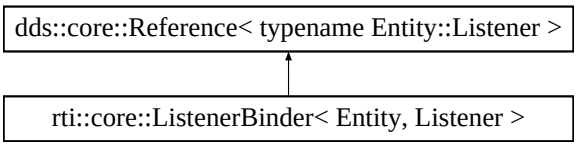
dds::core::cond::WaitSet (p. 2617), **dds::core::cond::Condition** (p. 948)

8.161 rti::core::ListenerBinder< Entity, Listener > Class Template Reference

<<***reference-type***>> *(*p. *351)* Automatically manages the association of an Entity and a **Listener** (p. 1637)

`#include <rti/core/ListenerBinder.hpp>`

Inheritance diagram for rti::core::ListenerBinder< Entity, Listener >:



Public Member Functions

- **Listener** * **get** ()

  (Deprecated) Use ***listener()*** *(*p. *1643)*
- const **Listener** * **get** () const

  (Deprecated) Use ***listener()*** *(*p. *1643)*
- **Listener** * **listener** ()

  Retrieves the listener.
- const **Listener** * **listener** () const

  Retrieves the listener.
- Entity **entity** () const

  Retrieves the Entity associated with the listener.

Related Functions

(Note that these are not member functions.)

- template<typename Entity , typename **Listener** >
  ListenerBinder< Entity, **Listener** > **bind_listener** (Entity **entity**, **Listener** *the_listener, **dds::core::status**↩
  ::StatusMask mask)

  Sets the listener and creates a ***ListenerBinder*** *(*p. *1641) that automatically resets it when all references go out of scope.*
- template<typename Entity , typename **Listener** >
  ListenerBinder< Entity, **Listener** > **bind_listener** (Entity **entity**, **Listener** *the_listener)

  Sets the listener and creates a ***ListenerBinder*** *(*p. *1641) that automatically resets it when all references go out of scope.*
- template<typename Entity , typename **Listener** >
  ListenerBinder< Entity, **Listener** > **bind_and_manage_listener** (Entity **entity**, **Listener** *the_listener, **dds::core::status::StatusMask** mask)

  Sets the listener and creates a ***ListenerBinder*** *(*p. *1641) that automatically resets and deletes it when all references go out of scope.*
- template<typename Entity , typename **Listener** >
  ListenerBinder< Entity, **Listener** > **bind_and_manage_listener** (Entity **entity**, **Listener** *the_listener)

  Sets the listener and creates a ***ListenerBinder*** *(*p. *1641) that automatically resets and deletes it when all references go out of scope.*

8.161.1 Detailed Description

template<typename Entity, typename Listener = typename Entity::Listener>
class rti::core::ListenerBinder< Entity, Listener >

<<***reference-type***>> *(p. 351)* Automatically manages the association of an Entity and a **Listener** (p. 1637)

[DEPRECATED] The use of **ListenerBinder** (p. 1641) is no longer required as long as the a `std::shared_ptr` is used to set an Entity's listener.

Note

A **ListenerBinder** (p. 1641) provides all the functions of a <<***reference-type***>> *(p. 351)* except **close()** (p. 1024) and retain().

Ties the association listener/Entity to the existence of references to this type–that is, the constructor sets the listener; when the last reference is destroyed, the entity's listener is unset. Depending on how this type is created, the listener may also be deleted:

- **`rti::core::bind_listener()`** (p. 708) creates a **ListenerBinder** (p. 1641) that doesn't own the listener and won't delete it
- **`rti::core::bind_and_manage_listener()`** (p. 709) creates a **ListenerBinder** (p. 1641) that owns the listener and will delete it.

The goal is to simplify the destruction of an Entity by the application, which otherwise would have to reset the listener or close the Entity explicitly.

Example:

```
class MyListener : public DataReaderListener<Foo> { ... };
void f()
{
    MyListener listener;
    dds::sub::DataReader<Foo> my_reader(subscriber, topic);
    {
        auto scoped_listener = rti::core::bind_listener(
                my_reader,
                listener,
                dds::core::status::StatusMask::data_available());
       // my_reader has a listener
       // ...
    } // After this, my_reader doesn't have a listener
} // my_reader destroyed; if we hadn't used a ListenerBinder,
  // my_reader would not have been destroyed
```

To create a **ListenerBinder** (p. 1641) use **rti::core::bind_listener()** (p. 708) or **rti::core::bind_and_manage_listener()** (p. 709). To retrieve the listener use **ListenerBinder::listener()** (p. 1643); to retrieve the entity, use **ListenerBinder**↩ **::entity()** (p. 1643).

If the Entity changes its listener while references to this **ListenerBinder** (p. 1641) still exist, the **ListenerBinder** (p. 1641) will not reset the listener.

Template Parameters

Entity	A subclass of **dds::core::Entity** (p. 1520)
Listener *(p. 1637)*	The listener type, by default `Entity::Listener`

See also

Reference types (p. 351), for an explanation of how reference types work and the reasons why an Entity may be retained.

8.161.2 Member Function Documentation

8.161.2.1 get() [1/2]

```
template<typename Entity , typename  Listener = typename Entity::Listener>
Listener *  rti::core::ListenerBinder< Entity,  Listener >::get ( )  [inline]
```

(Deprecated) Use **listener()** (p. 1643)

8.161.2.2 get() [2/2]

```
template<typename Entity , typename  Listener = typename Entity::Listener>
const  Listener *  rti::core::ListenerBinder< Entity,  Listener >::get ( ) const  [inline]
```

(Deprecated) Use **listener()** (p. 1643)

8.161.2.3 listener() [1/2]

```
template<typename Entity , typename  Listener = typename Entity::Listener>
Listener *  rti::core::ListenerBinder< Entity,  Listener >::listener ( )  [inline]
```

Retrieves the listener.

Returns

The listener.

8.161.2.4 listener() [2/2]

```
template<typename Entity , typename  Listener = typename Entity::Listener>
const  Listener *  rti::core::ListenerBinder< Entity,  Listener >::listener ( ) const  [inline]
```

Retrieves the listener.

8.161.2.5 entity()

```
template<typename Entity , typename  Listener = typename Entity::Listener>
Entity  rti::core::ListenerBinder< Entity,  Listener >::entity ( ) const  [inline]
```

Retrieves the Entity associated with the listener.

8.161.3 Friends And Related Function Documentation

8.161.3.1 bind_listener() [1/2]

```
template<typename Entity , typename  Listener >
ListenerBinder< Entity,  Listener > bind_listener (
             Entity entity,
              Listener * the_listener,
              dds::core::status::StatusMask mask )  [related]
```

Sets the listener and creates a **ListenerBinder** (p. 1641) that automatically resets it when all references go out of scope.

[DEPRECATED] The use of **ListenerBinder** (p. 1641) is no longer required as long as the a `std::shared_ptr` is used to set an Entity's listener.

The **ListenerBinder** (p. 1641) created from this function does not delete the listener, it simply sets the listener back to NULL in the Entity. The application is responsible for the life-cycle of the listener.

To also delete the listener when all references go out of scope, see **bind_and_manage_listener()** (p. 1644).

```
MyListener my_listener;
dds::sub::DataReader<Foo> reader(subscriber, topic);
// reader will have no listener after scoped_listener (and any other
// references created from scoped_listener) go out of scope--note that
// my_listener is a stack variable so scoped_listener should not delete it
auto scoped_listener = bind_listener(
    reader, &my_listener, dds::core::status::StatusMask::data_available());
```

8.161.3.2 bind_listener() [2/2]

```
template<typename Entity , typename  Listener >
ListenerBinder< Entity,  Listener > bind_listener (
             Entity entity,
              Listener * the_listener )  [related]
```

Sets the listener and creates a **ListenerBinder** (p. 1641) that automatically resets it when all references go out of scope.

[DEPRECATED] The use of **ListenerBinder** (p. 1641) is no longer required as long as the a `std::shared_ptr` is used to set an Entity's listener.

This overload works for listeners that don't use a mask

See also

bind_listener(Entity, Listener*, dds::core::status::StatusMask) (p. 1644)

8.161.3.3 bind_and_manage_listener() [1/2]

```
template<typename Entity , typename  Listener >
ListenerBinder< Entity,  Listener > bind_and_manage_listener (
             Entity entity,
              Listener * the_listener,
              dds::core::status::StatusMask mask )  [related]
```

Sets the listener and creates a **ListenerBinder** (p. 1641) that automatically resets and deletes it when all references go out of scope.

[DEPRECATED] The use of **ListenerBinder** (p. 1641) is no longer required as long as the a `std::shared_ptr` is used to set an Entity's listener.

Example:

```
dds::sub::DataReader<Foo> reader(subscriber, topic);
// The instance of MyListener we are creating and attaching to reader will be
// unset and deleted when scoped_listener (and any other references create
from
// scoped_listener) go out of scope
auto scoped_listener = bind_and_manage_listener(
    reader, new MyListener(),
dds::core::status::StatusMask::data_available());
```

See also

bind_listener() (p. 1644)

8.161.3.4 bind_and_manage_listener() [2/2]

```
template<typename Entity , typename  Listener >
ListenerBinder< Entity,  Listener > bind_and_manage_listener (
             Entity entity,
              Listener * the_listener )  [related]
```

Sets the listener and creates a **ListenerBinder** (p. 1641) that automatically resets and deletes it when all references go out of scope.

[DEPRECATED] The use of **ListenerBinder** (p. 1641) is no longer required as long as the a `std::shared_ptr` is used to set an Entity's listener.

This overload works for listeners that don't use a mask

See also

bind_and_manage_listener(Entity, Listener*, dds::core::status::StatusMask) (p. 1644)

8.162 dds::core::policy::Liveliness Class Reference

Specifies and configures the mechanism that allows **dds::sub::DataReader** (p. 976)'s to detect when **dds::pub::↩ DataWriter** (p. 1138)'s become disconnected.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **Liveliness** ()

  Creates an automatic liveliness policy with infinite lease duration.
- **Liveliness** (**dds::core::policy::LivelinessKind** the_kind, const **dds::core::Duration** &the_lease_duration)

  Creates an instance with the specified liveliness kind and lease duration.
- **Liveliness** & **kind** (**dds::core::policy::LivelinessKind** the_kind)

  Sets the liveliness kind.
- **dds::core::policy::LivelinessKind kind** () const

  Getter (see the setter with the same name)
- **Liveliness** & **lease_duration** (const **dds::core::Duration** &the_lease_duration)

  *Sets the duration within which a **dds::core::Entity** (p. 1520) must be asserted or else it is considered not alive.*
- **dds::core::Duration lease_duration** () const

  Getter (see the setter with the same name)
- **dds::core::policy::Liveliness** & **assertions_per_lease_duration** (int32_t value)

  *<<**extension**>> (p. 116) The number of assertions to send during the lease duration*
- int32_t **assertions_per_lease_duration** () const

  *<<**extension**>> (p. 116) Getter (see setter with the same name)*

Static Public Member Functions

- static **Liveliness Automatic** ()

  *Creates a **Liveliness** (p. 1645) instance with LivelinessKind::AUTOMATIC.*
- static **Liveliness ManualByParticipant** (const **dds::core::Duration** &lease= **dds::core::Duration::infinite**())

  *Creates a **Liveliness** (p. 1645) instance with LivelinessKind::MANUAL_BY_PARTICIPANT and the specified lease duration (infinite by default)*
- static **Liveliness ManualByTopic** (const **dds::core::Duration** &lease= **dds::core::Duration::infinite**())

  *Creates a **Liveliness** (p. 1645) instance with LivelinessKind::MANUAL_BY_TOPIC and the specified lease duration (infinite by default)*

8.162.1 Detailed Description

Specifies and configures the mechanism that allows **dds::sub::DataReader** (p. 976)'s to detect when **dds::pub::↩ DataWriter** (p. 1138)'s become disconnected.

The liveliness status of a **dds::pub::DataWriter** (p. 1138) is used to maintain instance ownership in combination with the setting of the **OWNERSHIP** (p. 538) policy. The application is also informed via **Listener** (p. 1637) when an **dds↩ ::pub::DataWriter** (p. 1138) is no longer alive.

A **dds::pub::DataWriter** (p. 1138) commits to signalling its liveliness at intervals not to exceed the **dds::core::policy↩ ::Liveliness::lease_duration** (p. 1649) configured on the **dds::pub::DataWriter** (p. 1138). The rate at which the **dds↩ ::pub::DataWriter** (p. 1138) will signal its liveliness is defined by **dds::core::policy::Liveliness::assertions_per_↩ lease_duration** (p. 1651).

The **dds::sub::DataReader** (p. 976) lease_duration specifies the maximum period at which matching DataWriters must have their liveliness asserted.

In addition, in the subscribing application Connext DDS uses an internal thread that wakes up at the period set by the DataReader's lease_duration to see if a DataWriter lease_duration has been violated.

Important: The DataReader lease_duration only configures how often a DataReader will check the liveliness of the matching DataWriters. The DataWriter lease_duration configures the period at which a DataWriter must assert its liveliness and is the value that the DataReader uses to determine if a matching DataWriter is alive or not.

Listeners are used to notify a **dds::sub::DataReader** (p. 976) of loss of liveliness and **dds::pub::DataWriter** (p. 1138) of violations to the liveliness contract. The on_liveliness_lost() callback is only called *once*, after the first time the lease_duration is exceeded (when the **dds::pub::DataWriter** (p. 1138) first loses liveliness).

This QoS policy can be used during system integration to ensure that applications have been coded to meet design specifications. It can also be used during runtime to detect when systems are performing outside of design specifications. Receiving applications can take appropriate actions in response to disconnected DataWriters.

Entity:

dds::topic::Topic (p. 2474), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Status:

dds::core::status::StatusMask::liveliness_lost() (p. 2383), **dds::core::status::LivelinessLostStatus** (p. 1655);
dds::core::status::StatusMask::liveliness_changed() (p. 2383), **dds::core::status::LivelinessChanged↩ Status** (p. 1652);
dds::core::status::StatusMask::requested_incompatible_qos() (p. 2381), **dds::core::status::StatusMask↩ ::offered_incompatible_qos()** (p. 2380)

Properties:

RxO (p. 514) = YES
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

8.162.2 Usage

This policy controls the mechanism and parameters used by RTI Connext to ensure that particular DataWriters on the network are still alive. The liveliness can also affect the ownership of a particular instance, as determined by the **OWNERSHIP** (p. 538) policy.

This policy has several settings to support both data types that are updated periodically as well as those that are changed sporadically. It also allows customisation for different application requirements in terms of the kinds of failures that will be detected by the liveliness mechanism.

The **dds::core::policy::LivelinessKind_def::AUTOMATIC** (p. 1655) liveliness setting is most appropriate for applications that only need to detect failures at the process-level, but not application-logic failures within a process. RTI Connext takes responsibility for renewing the leases at the required rates and thus, as long as the local process where a **dds↩ ::domain::DomainParticipant** (p. 1326) is running and the link connecting it to remote participants remains connected, the entities within the **dds::domain::DomainParticipant** (p. 1326) will be considered alive. This requires the lowest overhead.

The manual settings (**dds::core::policy::LivelinessKind_def::MANUAL_BY_PARTICIPANT** (p. 1655), **dds::core::policy::LivelinessKind_def::MANUAL_BY_TOPIC** (p. 1655)) require the application on the publishing side to periodically assert the liveliness before the lease expires to indicate the corresponding **dds::core::Entity** (p. 1520) is still alive. The action can be explicit by calling the **dds::pub::DataWriter::assert_liveliness** (p. 1172) operation or implicit by writing some data.

The two possible manual settings control the granularity at which the application must assert liveliness.

- The setting **dds::core::policy::LivelinessKind_def::MANUAL_BY_PARTICIPANT** (p. 1655) requires only that one **dds::core::Entity** (p. 1520) within a participant is asserted to be alive to deduce all other **dds::core::Entity** (p. 1520) objects within the same **dds::domain::DomainParticipant** (p. 1326) are also alive.

- The setting **dds::core::policy::LivelinessKind_def::MANUAL_BY_TOPIC** (p. 1655) requires that at least one instance within the **dds::pub::DataWriter** (p. 1138) is asserted.

Changes in **LIVELINESS** (p. 535) must be detected by the Service with a time-granularity greater or equal to the **dds::core::policy::Liveliness::lease_duration** (p. 1649). This ensures that the value of the **dds::core::status::LivelinessChangedStatus** (p. 1652) is updated at least once during each lease_duration and the related Listeners and **dds::core::cond::WaitSet** (p. 2617) s are notified within a lease_duration from the time the **LIVELINESS** (p. 535) changed.

8.162.3 Compatibility

The value offered is considered compatible with the value requested if and only if the following conditions are met:

- the inequality *offered kind >= requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of **dds::core::policy::LivelinessKind_def** (p. 1654) kind are considered ordered such that: **dds::core::policy::LivelinessKind_def::AUTOMATIC** (p. 1655) < **dds::core::policy::LivelinessKind_def::MANUAL_BY_PARTICIPANT** (p. 1655) < **dds::core::policy::LivelinessKind_def::MANUAL_BY_TOPIC** (p. 1655).

- the inequality *offered* `lease_duration` *<= requested* `lease_duration` evaluates to true.

See also

Relationship between registration, liveliness and ownership (p. 1902)

8.162.4 Constructor & Destructor Documentation

8.162.4.1 Liveliness() [1/2]

```
dds::core::policy::Liveliness::Liveliness ( )  [inline]
```

Creates an automatic liveliness policy with infinite lease duration.

8.162.4.2 Liveliness() [2/2]

```
dds::core::policy::Liveliness::Liveliness (
             dds::core::policy::LivelinessKind the_kind,
            const  dds::core::Duration & the_lease_duration )  [inline]
```

Creates an instance with the specified liveliness kind and lease duration.

8.162.5 Member Function Documentation

8.162.5.1 kind() [1/2]

```
Liveliness & dds::core::policy::Liveliness::kind (
             dds::core::policy::LivelinessKind the_kind )  [inline]
```

Sets the liveliness kind.

[default] dds::core::policy::LivelinessKind_def::AUTOMATIC (p. 1655)

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Matching** (p. 293), **Liveliness** (p. 199)

8.162.5.2 kind() [2/2]

```
dds::core::policy::LivelinessKind dds::core::policy::Liveliness::kind ( ) const  [inline]
```

Getter (see the setter with the same name)

8.162.5.3 lease_duration() [1/2]

```
Liveliness & dds::core::policy::Liveliness::lease_duration (
            const  dds::core::Duration & the_lease_duration )  [inline]
```

Sets the duration within which a **dds::core::Entity** (p. 1520) must be asserted or else it is considered not alive.

*<<**basic**>> (p. 104) <<**spec**>> (p. 112)* The duration within which a **dds::pub::DataWriter** (p. 1138) must be asserted, or else it is assumed to be not alive.

For a DataWriter, the lease_duration specifies a timeout by which liveliness must be asserted for the DataWriter, or the DataWriter will be considered inactive or not alive.

For a DataReader, the lease_duration specifies the maximum period at which the DataReader will check to see if the matching DataWriters are still alive according to the DataWriters lease_duration value.

Important: The DataWriter lease_duration is the value that the DataReader uses to determine if a matching DataWriter is alive or not.

Writer-side filtering is disabled if the writer's lease_duration is set to a finite value.

[default] dds::core::Duration::infinite() (p. 1451)

[range] [0,1 year] or **dds::core::Duration::infinite()** (p. 1451)

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Matching** (p. 293), **Liveliness** (p. 199)

8.162.5.4 lease_duration() [2/2]

```
dds::core::Duration dds::core::policy::Liveliness::lease_duration ( ) const  [inline]
```

Getter (see the setter with the same name)

8.162.5.5 Automatic()

```
static  Liveliness dds::core::policy::Liveliness::Automatic ( )  [inline], [static]
```

Creates a **Liveliness** (p. 1645) instance with LivelinessKind::AUTOMATIC.

8.162.5.6 ManualByParticipant()

```
static Liveliness dds::core::policy::Liveliness::ManualByParticipant (
            const dds::core::Duration & lease = dds::core::Duration::infinite() ) [inline],
[static]
```

Creates a **Liveliness** (p. 1645) instance with LivelinessKind::MANUAL_BY_PARTICIPANT and the specified lease duration (infinite by default)

8.162.5.7 ManualByTopic()

```
static Liveliness dds::core::policy::Liveliness::ManualByTopic (
            const dds::core::Duration & lease = dds::core::Duration::infinite() ) [inline],
[static]
```

Creates a **Liveliness** (p. 1645) instance with LivelinessKind::MANUAL_BY_TOPIC and the specified lease duration (infinite by default)

8.162.5.8 assertions_per_lease_duration() [1/2]

```
dds::core::policy::Liveliness & assertions_per_lease_duration (
            int32_t value )
```

<<***extension***>> *(p. 116)* The number of assertions to send during the lease duration

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This field only applies to a **dds::pub::DataWriter** (p. 1138) and is not considered during QoS compatibility checks.

The default value is 3. A higher value will make the liveliness mechanism more robust against packet losses, but it will also increase the network traffic.

[default] 3

[range] [2, 100 million]

Categories

Immutable (p. 243), **Liveliness** (p. 199)

8.162.5.9 assertions_per_lease_duration() [2/2]

```
int32_t assertions_per_lease_duration ( ) const
```

<<***extension***>> *(*p. *116)* Getter (see setter with the same name)

8.163 dds::core::status::LivelinessChangedStatus Class Reference

Information about the status **dds::core::status::StatusMask::liveliness_changed()** (p. 2383)

```
#include <TStatus.hpp>
```

Public Member Functions

- int32_t **alive_count** () const

  The total count of currently alive ***dds::pub::DataWriter*** *(*p. *1138) entities that write the* ***dds::topic::Topic*** *(*p. *2474) that this* ***dds::sub::DataReader*** *(*p. *976) reads.*
- int32_t **not_alive_count** () const

  The total count of currently not_alive ***dds::pub::DataWriter*** *(*p. *1138) entities that write the* ***dds::topic::Topic*** *(*p. *2474) that this* ***dds::sub::DataReader*** *(*p. *976) reads.*
- int32_t **alive_count_change** () const

  The change in the alive_count since the last time the listener was called or the status was read.
- int32_t **not_alive_count_change** () const

  The change in the not_alive_count since the last time the listener was called or the status was read.
- const **dds::core::InstanceHandle** **last_publication_handle** () const

  This ***InstanceHandle*** *(*p. *1612) can be used to look up which remote* ***dds::pub::DataWriter*** *(*p. *1138) was the last to cause this DataReader's status to change, using* ***dds::sub::DataReader::matched_publication_data*** *(*p. *1033).*

8.163.1 Detailed Description

Information about the status **dds::core::status::StatusMask::liveliness_changed()** (p. 2383)

The **dds::sub::DataReaderListener::on_liveliness_changed** (p. 1057) callback may be invoked for the following reasons:

- The liveliness of any **dds::pub::DataWriter** (p. 1138) matching this DataReader (as defined by the **dds::core::policy::LivelinessKind_def** (p. 1654) setting) is lost.
- A DataWriter's liveliness is recovered after being lost.
- A new matching DataWriter has been discovered.
- A matching DataWriter has been deleted.
- A QoS Policy has changed such that a DataWriter that matched this DataReader before no longer matches (such as a change to the **dds::core::policy::Partition** (p. 1923)). In this case, RTI Connext will no longer keep track of the DataWriter's liveliness. Furthermore, consider two scenarios:

  - DataWriter was alive when it and DataReader stopped matching: **dds::core::status::Liveliness↩ ChangedStatus::alive_count** (p. 1653) will decrease (since there's one less matching alive DataWriter) and **dds::core::status::LivelinessChangedStatus::not_alive_count** (p. 1653) will remain the same (since the DataWriter is still alive).
  - DataWriter was not alive when it and DataReader stopped matching: **dds::core::status::Liveliness↩ ChangedStatus::alive_count** (p. 1653) will remain the same (since the matching DataWriter was not alive) and **dds::core::status::LivelinessChangedStatus::not_alive_count** (p. 1653) will decrease (since there's one less not-alive matching DataWriter).

    Note: There are several ways that a DataWriter and DataReader can become incompatible after the Data↩ Writer has lost liveliness. For example, when the **dds::core::policy::LivelinessKind_def** (p. 1654) is set to DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS, it is possible that the DataWriter has not asserted its liveliness in a timely manner, and then a QoS change occurs on the DataWriter or DataReader that makes the entities incompatible.

- A QoS Policy (such as the **dds::core::policy::Partition** (p. 1923)) has changed such that a DataWriter that was unmatched with the DataReader now matches.

8.163.2 Member Function Documentation

8.163.2.1 alive_count()

```
int32_t dds::core::status::LivelinessChangedStatus::alive_count ( ) const  [inline]
```

The total count of currently alive **dds::pub::DataWriter** (p. 1138) entities that write the **dds::topic::Topic** (p. 2474) that this **dds::sub::DataReader** (p. 976) reads.

8.163.2.2 not_alive_count()

```
int32_t dds::core::status::LivelinessChangedStatus::not_alive_count ( ) const  [inline]
```

The total count of currently not_alive **dds::pub::DataWriter** (p. 1138) entities that write the **dds::topic::Topic** (p. 2474) that this **dds::sub::DataReader** (p. 976) reads.

8.163.2.3 alive_count_change()

```
int32_t dds::core::status::LivelinessChangedStatus::alive_count_change ( ) const  [inline]
```

The change in the alive_count since the last time the listener was called or the status was read.

8.163.2.4 not_alive_count_change()

```
int32_t dds::core::status::LivelinessChangedStatus::not_alive_count_change ( ) const  [inline]
```

The change in the not_alive_count since the last time the listener was called or the status was read.

Note that a positive not_alive_count_change means one of the following:

- The DomainParticipant containing the matched DataWriter has lost liveliness or has been deleted.
- The matched DataWriter has lost liveliness or has been deleted.

8.163.2.5 last_publication_handle()

```
const  dds::core::InstanceHandle dds::core::status::LivelinessChangedStatus::last_publication_↩
handle ( ) const  [inline]
```

This **InstanceHandle** (p. 1612) can be used to look up which remote **dds::pub::DataWriter** (p. 1138) was the last to cause this DataReader's status to change, using **dds::sub::DataReader::matched_publication_data** (p. 1033).

It's possible that the DataWriter has been purged from the discovery database. (See the "Discovery Overview" section of the User's Manual.) If so, the **dds::sub::DataReader::matched_publication_data** (p. 1033) method will not be able to return information about the DataWriter. In this case, the only way to get information about the lost DataWriter is if you cached the information previously.

8.164 dds::core::policy::LivelinessKind_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) LivelinessKind.

```
#include <PolicyKind.hpp>
```

Public Types

- enum **type** {
  AUTOMATIC ,
  MANUAL_BY_PARTICIPANT ,
  MANUAL_BY_TOPIC }
  The underlying enum type.

8.164.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) LivelinessKind.

8.164.2 Member Enumeration Documentation

8.164.2.1 type

```
enum  dds::core::policy::LivelinessKind_def::type
```

The underlying enum type.

Enumerator

AUTOMATIC	**[default]** The infrastructure will automatically signal liveliness for the **dds::pub::DataWriter** (p. 1138) (s) at least as often as required by the **dds::pub::DataWriter** (p. 1138) (S) `lease_duration`. A **dds::pub::DataWriter** (p. 1138) with this setting does not need to take any specific action in order to be considered 'alive.' The **dds::pub::DataWriter** (p. 1138) is only 'not alive' when the participant to which it belongs terminates (gracefully or not), or when there is a network problem that prevents the current participant from contacting that remote participant.
MANUAL_BY_PARTICIPANT	RTI Connext will assume that as long as at least one **dds::pub::DataWriter** (p. 1138) belonging to the **dds::domain::DomainParticipant** (p. 1326) (or the **dds::domain::DomainParticipant** (p. 1326) itself) has asserted its liveliness, then the other DataWriters belonging to that same **dds::domain::DomainParticipant** (p. 1326) are also alive. The user application takes responsibility to signal liveliness to RTI Connext either by calling **dds::domain::DomainParticipant::assert_liveliness** (p. 1338), or by calling **dds::pub::DataWriter::assert_liveliness** (p. 1172), or **dds::pub::DataWriter::write()** (p. 1148) on any **dds::pub::DataWriter** (p. 1138) belonging to the **dds::domain::DomainParticipant** (p. 1326).
MANUAL_BY_TOPIC	RTI Connext will only assume liveliness of the **dds::pub::DataWriter** (p. 1138) if the application has asserted liveliness of that **dds::pub::DataWriter** (p. 1138) itself. The user application takes responsibility to signal liveliness to RTI Connext using the **dds::pub::DataWriter::assert_liveliness** (p. 1172) method, or by writing some data.

8.165 dds::core::status::LivelinessLostStatus Class Reference

Information about the status **dds::core::status::StatusMask::liveliness_lost()** (p. 2383)

`#include <TStatus.hpp>`

Public Member Functions

- int32_t **total_count** () const

  *Total cumulative number of times that a previously-alive **dds::pub::DataWriter** (p. 1138) became not alive due to a failure to to actively signal its liveliness within the offered liveliness period.*
- int32_t **total_count_change** () const

  The incremental changes in total_count since the last time the listener was called or the status was read.

8.165.1 Detailed Description

Information about the status **dds::core::status::StatusMask::liveliness_lost()** (p. 2383)

Entity:

dds::pub::DataWriter (p. 1138)

Listener:

dds::pub::DataWriterListener (p. 1202)

The liveliness that the **dds::pub::DataWriter** (p. 1138) has committed through its **dds::core::policy::Liveliness** (p. 1645) was not respected; thus **dds::sub::DataReader** (p. 976) entities will consider the **dds::pub::DataWriter** (p. 1138) as no longer "alive/active".

8.165.2 Member Function Documentation

8.165.2.1 total_count()

```
int32_t dds::core::status::LivelinessLostStatus::total_count ( ) const  [inline]
```

Total cumulative number of times that a previously-alive **dds::pub::DataWriter** (p. 1138) became not alive due to a failure to to actively signal its liveliness within the offered liveliness period.

This count does not change when an already not alive **dds::pub::DataWriter** (p. 1138) simply remains not alive for another liveliness period.

8.165.2.2 total_count_change()

```
int32_t dds::core::status::LivelinessLostStatus::total_count_change ( ) const  [inline]
```

The incremental changes in total_count since the last time the listener was called or the status was read.

8.166 rti::core::xtypes::LoanedDynamicData Class Reference

<<***move-only-type***>> *(p. 353)* Gives temporary access to a member of another DynamicData object.

```
#include <DynamicDataImpl.hpp>
```

Public Member Functions

- ~**LoanedDynamicData** ()

  Returns the loan if it has not been returned with ***return_loan()*** (p. *1658)*
- void **return_loan** ()

  Explicitly returns the loan.
- DynamicData & **get** ()

  Obtains the loaned DynamicData object representing a member of DynamicData object.
- const DynamicData & **get** () const

  Obtains the loaned DynamicData object representing a member of DynamicData object.
- **operator DynamicData &** ()

  Conversion to DynamicData for convenience.
- **operator const DynamicData &** () const

  Conversion to DynamicData for convenience.
- **LoanedDynamicData** (**LoanedDynamicData** &&other) OMG_NOEXCEPT

  *<<**C++11**>> (p. 353) Move constructor.*
- **LoanedDynamicData** & **operator=** (**LoanedDynamicData** &&other) OMG_NOEXCEPT

  *<<**C++11**>> (p. 353) Move-assignment operator.*

8.166.1 Detailed Description

*<<**move-only-type**>> (p. 353)* Gives temporary access to a member of another DynamicData object.

This type can only be instantiated through **dds::core::xtypes::DynamicData::loan_value** (p. 1475). This type can't be copied. It can be moved *<<**C++11**>> (p. 353)*.

An instance of this type contains a direct reference to a member of another DynamicData object, retrieved with **get()** (p. 1658). That object can be both inspected or modified and has to be eventually returned.

There are three ways to return the loan:

1. Letting the destructor do it
2. Calling **return_loan()** (p. 1658)
3. Reusing this object to obtain another loan (see **dds::core::xtypes::DynamicData::loan_value(LoanedDynamicData&,uint32_t)** (p. 1477))

See also

dds::core::xtypes::DynamicData::loan_value (p. 1475)

Using DynamicData (p. 611)

8.166.2 Constructor & Destructor Documentation

8.166.2.1 ~LoanedDynamicData()

```
rti::core::xtypes::LoanedDynamicData::~LoanedDynamicData ( )
```

Returns the loan if it has not been returned with **return_loan()** (p. 1658)

8.166.2.2 LoanedDynamicData()

```
rti::core::xtypes::LoanedDynamicData::LoanedDynamicData (
            LoanedDynamicData && other )  [inline]
```

<<***C++11***>> *(p. 353)* Move constructor.

A **LoanedDynamicData** (p. 1656) can only be moved, not copied.

8.166.3 Member Function Documentation

8.166.3.1 return_loan()

```
void rti::core::xtypes::LoanedDynamicData::return_loan ( )
```

Explicitly returns the loan.

8.166.3.2 get() [1/2]

```
DynamicData & rti::core::xtypes::LoanedDynamicData::get ( )  [inline]
```

Obtains the loaned DynamicData object representing a member of DynamicData object.

The object can be modified and the changes will take effect in the DynamicData object that contains this one.

8.166.3.3 get() [2/2]

```
const DynamicData & rti::core::xtypes::LoanedDynamicData::get ( ) const  [inline]
```

Obtains the loaned DynamicData object representing a member of DynamicData object.

8.166.3.4 operator DynamicData &()

```
rti::core::xtypes::LoanedDynamicData::operator DynamicData & ( )  [inline]
```

Conversion to DynamicData for convenience.

8.166.3.5 operator const DynamicData &()

```
rti::core::xtypes::LoanedDynamicData::operator const DynamicData & ( ) const  [inline]
```

Conversion to DynamicData for convenience.

8.166.3.6 operator=()

```
LoanedDynamicData & rti::core::xtypes::LoanedDynamicData::operator= (
            LoanedDynamicData && other )  [inline]
```

<<***C++11***>> *(*p. *353)* Move-assignment operator.

Since copying a **LoanedDynamicData** (p. 1656) is disabled, only the move-assignment operator is allowed.

This operation returns the loan that `*this` currently holds.

For example:

```
LoanedDynamicData member = sample.loan_value(1);
member = sample.loan_value(2);
```

See also

dds::core::xtypes::DynamicData::loan_value(LoanedDynamicData&, const std::string&) (p. 1476)

8.167 rti::sub::LoanedSample< T > Class Template Reference

The element type of a **dds::sub::LoanedSamples** (p. 1663) collection.

```
#include <LoanedSample.hpp>
```

Public Types

- typedef T **DataType**

  The data type.
- typedef **dds::sub::SampleInfo** **InfoType**

  dds::sub::SampleInfo *(*p. *2285)*

Public Member Functions

- const **DataType** & **data** () const
  Gets the data.
- const **InfoType** & **info** () const
  Gets the sample info.
- **operator const DataType &** () const
  Allows implicit conversion to the data type.
- bool **operator==** (const **LoanedSample** &other) const
  Compares the data and info.

Related Functions

(Note that these are not member functions.)

- template<typename T >
  dds::sub::Sample< T > **copy_to_sample** (const **rti::sub::LoanedSample**< T > &ls)
  *Copies the contents of a **rti::sub::LoanedSample** (p. 1659) into a **dds::sub::Sample** (p. 2270).*
- template<typename T >
  std::ostream & **operator**<< (std::ostream &out, const **LoanedSample**< T > &sample)
  Calls the operator on the data or prints [invalid data].

8.167.1 Detailed Description

template<typename T>
class rti::sub::LoanedSample< T >

The element type of a **dds::sub::LoanedSamples** (p. 1663) collection.

This class encapsulates loaned, read-only data and SampleInfo from a DataReader.

LoanedSample (p. 1659) instances are always the element of a **dds::sub::LoanedSamples** (p. 1663) collection and have to be returned through that collection. A **LoanedSample** (p. 1659) instance is a lightweight handle to data owned by the DataReader from where you received a LoanedSamples collection.

This type is not exactly a reference type, value type or move-only type. Copying a **LoanedSample** (p. 1659) simply creates a new handle to the same loaned data.

The difference between **LoanedSample** (p. 1659) and **dds::sub::Sample** (p. 2270) is that the latter is a **value type** (p. 350) that the application owns. A Sample can be constructed by copying the data and meta-data referenced by a **LoanedSample** (p. 1659).

In most cases the only thing applications care about is that the elements of a **dds::sub::LoanedSamples** (p. 1663) collection have two methods, **data()** (p. 1661) and **info()** (p. 1661). For example:

```
dds::sub::LoanedSamples<Foo> samples = reader.take();
for (auto sample : samples) {
    if (sample.info().valid()) {
        std::cout « sample.data() « std::endl;
    }
}
```

See also

dds::sub::LoanedSamples (p. 1663)

8.167.2 Member Typedef Documentation

8.167.2.1 DataType

```
template<typename T >
typedef T  rti::sub::LoanedSample< T >::DataType
```

The data type.

8.167.2.2 InfoType

```
template<typename T >
typedef  dds::sub::SampleInfo  rti::sub::LoanedSample< T >::InfoType
```

dds::sub::SampleInfo (p. 2285)

8.167.3 Member Function Documentation

8.167.3.1 data()

```
template<typename T >
const  DataType &  rti::sub::LoanedSample< T >::data ( ) const  [inline]
```

Gets the data.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !info().valid().

References **rti::sub::LoanedSample< T >::info()**.

Referenced by **rti::sub::LoanedSample< T >::operator const DataType &()**, **rti::sub::LoanedSample< T >::operator<<()**, and **rti::sub::LoanedSample< T >::operator==()**.

8.167.3.2 info()

```
template<typename T >
const  InfoType &  rti::sub::LoanedSample< T >::info ( ) const  [inline]
```

Gets the sample info.

Referenced by **rti::sub::LoanedSample< T >::data()**, **rti::request::IsReplyRelatedPredicate< T >::operator()()**, **rti::sub::LoanedSample< T >::operator<<()**, and **rti::sub::LoanedSample< T >::operator==()**.

8.167.3.3 operator const DataType &()

```
template<typename T >
rti::sub::LoanedSample< T >::operator const  DataType & ( ) const  [inline]
```

Allows implicit conversion to the data type.

One use of this conversion operator is to simplify the usage of generic algorithms that iterate on a LoanedSamples collection.

For example, the following example copies all the data of in a LoanedSamples collection into a vector of the data type. The call to `std::copy` works as-is thanks to this conversion.

```
LoanedSamples<KeyedType> samples = reader.take();
std::vector<KeyedType> data_vector;
std::copy(samples.begin(), samples.end(), std::back_inserter(data_vector));
```

Note: the example above may throw **dds::core::PreconditionNotMetError** (p. 1941) if any of the samples has invalid data. See **valid_data()** (p. 766) to iterate only over samples with valid data.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if `!info`().valid().

References **rti::sub::LoanedSample< T >::data()**.

8.167.3.4 operator==()

```
template<typename T >
bool  rti::sub::LoanedSample< T >::operator== (
            const  LoanedSample< T > & other ) const  [inline]
```

Compares the data and info.

References **rti::sub::LoanedSample< T >::data()**, and **rti::sub::LoanedSample< T >::info()**.

8.167.4 Friends And Related Function Documentation

8.167.4.1 copy_to_sample()

```
template<typename T >
dds::sub::Sample< T > copy_to_sample (
            const  rti::sub::LoanedSample< T > & ls )  [related]
```

Copies the contents of a **rti::sub::LoanedSample** (p. 1659) into a **dds::sub::Sample** (p. 2270).

Example:

```
dds::sub::LoanedSamples<Foo> samples = reader.take();
std::vector<Sample<Foo> > sample_vector;
std::transform(
    rti::sub::valid_data(samples.begin()),
    rti::sub::valid_data(samples.end()),
    std::back_inserter(sample_vector),
    rti::sub::copy_to_sample<Foo>);
```

8.167.4.2 operator<<()

```
template<typename T >
std::ostream & operator<< (
            std::ostream & out,
            const  LoanedSample< T > & sample )  [related]
```

Calls the operator on the data or prints [invalid data].

References **rti::sub::LoanedSample< T >::data()**, **rti::sub::LoanedSample< T >::info()**, and **dds::sub::SampleInfo::valid()**.

8.168 dds::sub::LoanedSamples< T > Class Template Reference

<<***move-only-type***>> *(p. 353)* Provides temporary access to a collection of samples (data and info) from a **DataReader** (p. 976).

```
#include <LoanedSamplesImpl.hpp>
```

Public Types

- typedef SampleIterator< T > **iterator**

  The iterator type.

Public Member Functions

- **LoanedSamples** ()=default

  *Creates an empty **LoanedSamples** (p. 1663) object.*
- ~**LoanedSamples** () noexcept

  *Automatically returns the loan to the **DataReader** (p. 976).*
- value_type **operator[ ]** (size_t index)

  Provides access to the underlying LoanedSample object in array-like syntax.
- unsigned int **length** () const

  Gets the number of samples in this collection.
- void **return_loan** ()

  *Returns the samples to the **DataReader** (p. 976).*
- bool **return_loan_noexcept** () noexcept

  *Returns the samples to the **DataReader** (p. 976) (noexcept version)*
- **iterator begin** ()

  Gets an iterator to the first sample.
- **iterator end** ()

  Gets an iterator to one past the last sample.
- const_iterator **begin** () const

  Gets an iterator to the first sample.
- const_iterator **end** () const

  Gets an iterator to one past the last sample.
- void **swap** (**LoanedSamples** &other) throw ()

  *Swap two **LoanedSamples** (p. 1663) containers.*
- **LoanedSamples** (**LoanedSamples** &&other)

  *<<**C++11**>> (p. 353) Moves the loan from an existing **LoanedSamples** (p. 1663) to a new one*

Related Functions

(Note that these are not member functions.)

- template<typename T >
  LoanedSamples< T > **move** (**LoanedSamples**< T > &ls) OMG_NOEXCEPT

  *Creates a new **LoanedSamples** (p. 1663) instance by moving the contents of an existing one.*
- template<typename T >
  LoanedSamples< T > **::iterator begin** (**LoanedSamples**< T > &ls)
- template<typename T >
  LoanedSamples< T >::const_iterator **begin** (const **LoanedSamples**< T > &ls)
- template<typename T >
  LoanedSamples< T > **::iterator end** (**LoanedSamples**< T > &ls)
- template<typename T >
  LoanedSamples< T >::const_iterator **end** (const **LoanedSamples**< T > &ls)
- template<typename T >
  void **swap** (**LoanedSamples**< T > &ls1, **LoanedSamples**< T > &ls2) throw()
- template<typename T >
  ValidLoanedSamples< T > **valid_data** (**LoanedSamples**< T > &&samples)

  *<<**C++11**>> (p. 353) <<**extension**>> (p. 354) Returns a collection that provides access only to samples with valid data*
- template<typename T >
  ValidSampleIterator< T > **valid_data** (const SampleIterator< T > &sample_iterator)

  *<<**extension**>> (p. 354) Returns an iterator that skips invalid samples*

8.168.1 Detailed Description

template<typename T>
class dds::sub::LoanedSamples< T >

<<***move-only-type***>> *(p. 353)* Provides temporary access to a collection of samples (data and info) from a **DataReader** (p. 976).

Template Parameters

T	The topic-type. It has to match the type of the **DataReader** (p. 976).

This STL-like container encapsulates a collection of loaned, read-only data samples (**data** (p. 1661) and **info** (p. 1661)) from a **DataReader** (p. 976).

To obtain a **LoanedSamples** (p. 1663) you need to call one of the read/take operations from a **DataReader** (p. 976). The samples have to be eventually returned to the **DataReader** (p. 976). The destructor takes care of that, and the **return_loan()** (p. 1667) function lets you do it explicitly if needed.

As a move-only type copying a **LoanedSamples** (p. 1663) is not allowed. If you want to have more than one reference to a collection of loaned sample, see **SharedSamples** (p. 2362). If you need to return a **LoanedSamples** (p. 1663) from a function or assign it to another variable, use `dds::core::move()` (or `std::move()` <<***C++11***>> *(p. 353)*).

Iterators and overloaded subscript operators let you access the samples in this container, which are of the type **rti::sub::LoanedSample** (p. 1659).

This code demonstrates how to access the info and data of each sample in a **DataReader** (p. 976):

```
auto samples = reader.take();
for (auto sample : samples) {
    if (sample.info().valid()) {
        std::cout « sample.data() « std::endl;
    }
}
```

See also

Reading data samples (p. 318) for more examples

Examples

Foo_subscriber.cxx.

8.168.2 Member Typedef Documentation

8.168.2.1 iterator

```
template<typename T >
typedef SampleIterator<T>  dds::sub::LoanedSamples< T >::iterator
```

The iterator type.

8.168.3 Constructor & Destructor Documentation

8.168.3.1 LoanedSamples() [1/2]

```
template<typename T >
dds::sub::LoanedSamples< T >::LoanedSamples ( )  [default]
```

Creates an empty **LoanedSamples** (p. 1663) object.

8.168.3.2 ~LoanedSamples()

```
template<typename T >
dds::sub::LoanedSamples< T >::~ LoanedSamples ( )  [inline], [noexcept]
```

Automatically returns the loan to the **DataReader** (p. 976).

See also

return_loan (p. 1667)

8.168.3.3 LoanedSamples() [2/2]

```
template<typename T >
dds::sub::LoanedSamples< T >::LoanedSamples (
             LoanedSamples< T > && other )  [inline]
```

<<***C++11***>> *(p. 353)* Moves the loan from an existing **LoanedSamples** (p. 1663) to a new one

References **dds::sub::LoanedSamples< T >::swap()**.

8.168.4 Member Function Documentation

8.168.4.1 operator[]()

```
template<typename T >
value_type  dds::sub::LoanedSamples< T >::operator[] (
             size_t index )  [inline]
```

Provides access to the underlying LoanedSample object in array-like syntax.

Parameters

index	The index of the **Sample** (p. 2270). Allowed values are from 0 to **length()** (p. 1667)-1.

Returns

A LoanedSample object that refers to data and **SampleInfo** (p. 2285) at the specified `index`.

8.168.4.2 length()

```
template<typename T >
unsigned int  dds::sub::LoanedSamples< T >::length ( ) const  [inline]
```

Gets the number of samples in this collection.

8.168.4.3 return_loan()

```
template<typename T >
void  dds::sub::LoanedSamples< T >::return_loan ( )  [inline]
```

Returns the samples to the **DataReader** (p. 976).

Note

Explicitly calling return_loan is optional, since the destructor does it implicitly.

This operation tells the **dds::sub::DataReader** (p. 976) that the application is done accessing the collection of samples.

It is not necessary for an application to return the loans immediately after the call to read or take. However, as these buffers correspond to internal resources, the application should not retain them indefinitely.

8.168.4.4 return_loan_noexcept()

```
template<typename T >
bool  dds::sub::LoanedSamples< T >::return_loan_noexcept ( )  [inline], [noexcept]
```

Returns the samples to the **DataReader** (p. 976) (noexcept version)

This operation has the same behavior as **return_loan()** (p. 1667) but returns false in case of error and doesn't throw exceptions.

8.168.4.5 begin() [1/2]

```
template<typename T >
iterator  dds::sub::LoanedSamples< T >::begin ( )  [inline]
```

Gets an iterator to the first sample.

Referenced by **dds::sub::LoanedSamples< T >::begin()**.

8.168.4.6 end() [1/2]

```
template<typename T >
iterator  dds::sub::LoanedSamples< T >::end ( )  [inline]
```

Gets an iterator to one past the last sample.

Referenced by **dds::sub::LoanedSamples< T >::end()**.

8.168.4.7 begin() [2/2]

```
template<typename T >
const_iterator  dds::sub::LoanedSamples< T >::begin ( ) const  [inline]
```

Gets an iterator to the first sample.

8.168.4.8 end() [2/2]

```
template<typename T >
const_iterator  dds::sub::LoanedSamples< T >::end ( ) const  [inline]
```

Gets an iterator to one past the last sample.

8.168.4.9 swap()

```
template<typename T >
void  dds::sub::LoanedSamples< T >::swap (
              LoanedSamples< T > & other ) throw ( )   [inline]
```

Swap two **LoanedSamples** (p. 1663) containers.

Referenced by **dds::sub::LoanedSamples< T >::LoanedSamples()**.

8.168.5 Friends And Related Function Documentation

8.168.5.1 move()

```
template<typename T >
LoanedSamples< T > move (
              LoanedSamples< T > & ls )  [related]
```

Creates a new **LoanedSamples** (p. 1663) instance by moving the contents of an existing one.

This function existed to provide move semantics before C++11, but is now deprecated in favor of std::move.

[DEPRECATED] Use std::move and the move constructor of **LoanedSamples** (p. 1663) instead. For example:

```
LoanedSamples<Foo> samples1 = reader.take();
LoanedSamples<Foo> samples2 = std::move(samples1);
```

Parameters

ls	The **LoanedSamples** (p. 1663) object that transfers its ownership of the contained samples into the returned object. After this call, `ls` is empty.

Returns

A new **LoanedSamples** (p. 1663) object, the new loan owner, with the same contents as `ls` had.

See also

LoanedSamples (p. 1663)

References **dds::sub::move()**.

8.168.5.2 begin() [1/2]

```
template<typename T >
LoanedSamples< T > ::iterator begin (
              LoanedSamples< T > & ls )  [related]
```

See also

LoanedSamples::begin() (p. 1667)

References **dds::sub::LoanedSamples< T >::begin()**.

8.168.5.3 begin() [2/2]

```
template<typename T >
LoanedSamples< T >::const_iterator begin (
            const  LoanedSamples< T > & ls )  [related]
```

See also

LoanedSamples::begin() (p. 1667)

References **dds::sub::LoanedSamples< T >::begin()**.

8.168.5.4 end() [1/2]

```
template<typename T >
LoanedSamples< T > ::iterator end (
             LoanedSamples< T > & ls )  [related]
```

See also

LoanedSamples::end() (p. 1668)

References **dds::sub::LoanedSamples< T >::end()**.

8.168.5.5 end() [2/2]

```
template<typename T >
LoanedSamples< T >::const_iterator end (
            const  LoanedSamples< T > & ls )  [related]
```

See also

LoanedSamples::end() (p. 1668)

References **dds::sub::LoanedSamples< T >::end()**.

8.168.5.6 swap()

```
template<typename T >
void swap (
              LoanedSamples< T > & ls1,
              LoanedSamples< T > & ls2 ) throw( )   [related]
```

See also

LoanedSamples::swap() (p. 1668)

8.168.5.7 valid_data() [1/2]

```
template<typename T >
ValidLoanedSamples< T > valid_data (
              LoanedSamples< T > && samples )  [related]
```

<<***C++11***>> *(p. 353)* <<***extension***>> *(p. 354)* Returns a collection that provides access only to samples with valid data

Template Parameters

T	The topic-type. It has to match the type of the **DataReader** (p. 976).

This function transforms a **LoanedSamples** (p. 1663) collection into another collection whose iterators only access valid-data samples, skipping any sample such that !sample.info().valid().

This operation is O(1) and will not copy the data samples or allocated any additional memory.

The typical way to use this function is to directly call it on the return value of a **read()** (p. 661)/take() operation and use it in a for-loop. For example:

```
auto valid_samples = rti::sub::valid_data(reader.read());
for (auto sample : valid_samples) {
    // no need to check sample.info().valid()
    std::cout « sample.data() « std::endl;
}
```

Parameters

samples	The collection of **LoanedSamples** (p. 1663) to transform into a ValidLoanedSamples. It must be an rvalue, so valid actual parameters are the result of one of the read/take operations:
	`auto vs = rti::sub::valid_data(reader.take());`
	Or an `std::move'd` existing collection:
	`auto ls = reader.take(); auto vs = rti::sub::valid_data(std::move(ls));`
	`// 'ls' is now invalid and can't be further used`

Returns

A forward-iterable collection that provides access only to samples with valid data. Note that this collection doesn't provide random access.

Postcondition

`samples` is invalid cannot be used after this call

See also

rti::sub::valid_data(const SampleIterator<T>&) (p. 767), which applies to an **iterator** (p. 1665) rather to the whole collection

Reading data samples (p. 318)

8.168.5.8 valid_data() [2/2]

```
template<typename T >
ValidSampleIterator< T > valid_data (
             const SampleIterator< T > & sample_iterator )  [related]
```

<<***extension***>> *(p. 354)* Returns an iterator that skips invalid samples

Given a regular sample iterator, this functions creates another iterator `it` that behaves exactly the same except that `it++` moves to the next valid sample (or to the end of the collection). That is, if `it` doesn't point to the end of the collection, `it->info.valid()` is always true.

This is useful when your application doesn't need to deal with samples containing meta-information only.

For example, the following code copies all the data in a **LoanedSamples** (p. 1663) collection skipping any invalid samples (otherwise, attempting to copy the data from an invalid sample would throw an exception, see rti::sub::Loaned↩ Sample::operator const DataType& ()).

```
dds::sub::LoanedSamples<KeyedType> samples = reader.take();
std::vector<KeyedType> data_vector;
std::copy(
     rti::sub::valid_data(samples.begin()),
     rti::sub::valid_data(samples.end()),
     std::back_inserter(data_vector));
```

Note that `valid_data`(samples.begin()) won't point to the first element if that element is not a valid sample.

A similar utility is the functor **rti::sub::IsValidData** (p. 1624).

See also

dds::sub::LoanedSamples (p. 1663)

rti::sub::IsValidData (p. 1624)

dds::sub::SampleInfo::valid() (p. 2289)

rti::sub::valid_data(LoanedSamples<T>&&) (p. 766), which applies to the whole collection instead of an **iterator** (p. 1665)

Reading data samples (p. 318)

8.169 rti::core::Locator Class Reference

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

```
#include "rti/core/Locator.hpp"
```

Public Member Functions

- **Locator** (const rti::core::LocatorKind::type the_kind, uint32_t the_port, const **dds::core::ByteSeq** &the_address)

  *Construct a **Locator** (p. 1673) with the provided kind, port and address.*
- rti::core::LocatorKind::type **kind** () const

  *Get the kind associated with this **Locator** (p. 1673).*
- **Locator** & **kind** (const rti::core::LocatorKind::type the_kind)

  *Set the kind associated with this **Locator** (p. 1673).*
- uint32_t **port** () const

  *Get the port number associated with this **Locator** (p. 1673).*
- **Locator** & **port** (uint32_t the_port)

  *Set the port number associated with this **Locator** (p. 1673).*
- **dds::core::ByteSeq** **address** () const

  *Get the IP address associated with this **Locator** (p. 1673).*
- **Locator** & **address** (const **dds::core::ByteSeq** &the_address)

  *Set the address associated with this **Locator** (p. 1673) object.*

Static Public Member Functions

- static **Locator** **Invalid** ()

  Construct an invalid locator.

Related Functions

(Note that these are not member functions.)

- typedef std::vector< **Locator** > **LocatorSeq**

  *A sequence of **rti::core::Locator** (p. 1673).*

8.169.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

8.169.2 Constructor & Destructor Documentation

8.169.2.1 Locator()

```
rti::core::Locator::Locator (
            const rti::core::LocatorKind::type the_kind,
            uint32_t the_port,
            const  dds::core::ByteSeq & the_address )  [inline]
```

Construct a **Locator** (p. 1673) with the provided kind, port and address.

8.169.3 Member Function Documentation

8.169.3.1 kind() [1/2]

```
rti::core::LocatorKind::type rti::core::Locator::kind ( ) const  [inline]
```

Get the kind associated with this **Locator** (p. 1673).

The kind of locator.

If the Locator_t kind is Locator_t::KIND_UDPv4, the address contains an IPv4 address. In this case, the leading 12 octets of the Locator_t::address must be zero. The last 4 octets of Locator_t::address are used to store the IPv4 address.

If the Locator_t kind is Locator_t::KIND_UDPv6, the address contains an IPv6 address. IPv6 addresses typically use a shorthand hexadecimal notation that maps one-to-one to the 16 octets in the Locator_t::address field.

See also

rti::core::LocatorKind_def (p. 1681)

8.169.3.2 kind() [2/2]

```
Locator & rti::core::Locator::kind (
            const rti::core::LocatorKind::type the_kind )  [inline]
```

Set the kind associated with this **Locator** (p. 1673).

See also

rti::core::LocatorKind_def (p. 1681)

8.169.3.3 port() [1/2]

```
uint32_t rti::core::Locator::port ( ) const  [inline]
```

Get the port number associated with this **Locator** (p. 1673).

8.169.3.4 port() [2/2]

```
Locator & rti::core::Locator::port (
            uint32_t the_port )  [inline]
```

Set the port number associated with this **Locator** (p. 1673).

8.169.3.5 address() [1/2]

```
dds::core::ByteSeq rti::core::Locator::address ( ) const  [inline]
```

Get the IP address associated with this **Locator** (p. 1673).

8.169.3.6 address() [2/2]

```
Locator & rti::core::Locator::address (
            const  dds::core::ByteSeq & the_address )  [inline]
```

Set the address associated with this **Locator** (p. 1673) object.

8.169.3.7 Invalid()

```
static  Locator rti::core::Locator::Invalid ( )  [inline], [static]
```

Construct an invalid locator.

Since this class doesn't provide a default constructor if you don't know the values of a **Locator** (p. 1673) at the moment of construction use this function explicitly to obtain an invalid object to be filled out.

8.169.4 Friends And Related Function Documentation

8.169.4.1 LocatorSeq

```
typedef std::vector< Locator>  LocatorSeq  [related]
```

A sequence of **rti::core::Locator** (p. 1673).

8.170 rti::core::policy::LocatorFilter Class Reference

<<***extension***>> *(p. 116)* Configures how the **dds::topic::PublicationBuiltinTopicData** (p. 1977) reports the configuration of a **MultiChannel** (p. 1749) DataWriter.

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Types

- typedef **rti::core::LocatorFilterElement Filter**

  *A **LocatorFilter** (p. 1676) policy is a collection of **LocatorFilterElement** (p. 1678).*
- typedef std::vector< **Filter** > **FilterSeq**

  A vector of Filter.

Public Member Functions

- **LocatorFilter** ()

  Creates the default policy.
- **LocatorFilter** (const **FilterSeq** &the_locator_filters, const std::string &the_filter_name= **rti::topic::stringmatch_filter_name**())

  Creates an instance with a sequence of filters and a filter name.
- **LocatorFilter** & **locator_filters** (const **FilterSeq** &the_locator_filters)

  Sets the locator filters.
- **FilterSeq locator_filters** () const

  Gets the locator filters.
- **LocatorFilter** & **filter_name** (const std::string &the_filter_name)

  Sets the filter name.
- std::string **filter_name** () const

  Gets the filter name.

8.170.1 Detailed Description

<<***extension***>> *(p. 116)* Configures how the **dds::topic::PublicationBuiltinTopicData** (p. 1977) reports the configuration of a **MultiChannel** (p. 1749) DataWriter.

Entity:

dds::topic::PublicationBuiltinTopicData (p. 1977)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.170.2 Member Typedef Documentation

8.170.2.1 Filter

```
typedef  rti::core::LocatorFilterElement  rti::core::policy::LocatorFilter::Filter
```

A **LocatorFilter** (p. 1676) policy is a collection of **LocatorFilterElement** (p. 1678).

8.170.2.2 FilterSeq

```
typedef std::vector< Filter>  rti::core::policy::LocatorFilter::FilterSeq
```

A vector of Filter.

8.170.3 Constructor & Destructor Documentation

8.170.3.1 LocatorFilter() [1/2]

```
rti::core::policy::LocatorFilter::LocatorFilter ( )  [inline]
```

Creates the default policy.

8.170.3.2 LocatorFilter() [2/2]

```
rti::core::policy::LocatorFilter::LocatorFilter (
            const  FilterSeq & the_locator_filters,
            const std::string & the_filter_name =  rti::topic::stringmatch_filter_name() )  [inline]
```

Creates an instance with a sequence of filters and a filter name.

8.170.4 Member Function Documentation

8.170.4.1 locator_filters() [1/2]

```
LocatorFilter & rti::core::policy::LocatorFilter::locator_filters (
            const  FilterSeq & the_locator_filters )
```

Sets the locator filters.

A sequence length of zero indicates the **rti::core::policy::MultiChannel** (p. 1749) is not in use.

[default] Empty sequence.

8.170.4.2 locator_filters() [2/2]

```
FilterSeq rti::core::policy::LocatorFilter::locator_filters ( ) const
```

Gets the locator filters.

8.170.4.3 filter_name() [1/2]

```
LocatorFilter & rti::core::policy::LocatorFilter::filter_name (
            const std::string & the_filter_name )
```

Sets the filter name.

The following builtin filters are supported: **rti::topic::sql_filter_name** (p. 49) and **rti::topic::stringmatch_filter_name** (p. 49).

[default] rti::topic::stringmatch_filter_name (p. 49)

References **rti::topic::stringmatch_filter_name()**.

8.170.4.4 filter_name() [2/2]

```
std::string rti::core::policy::LocatorFilter::filter_name ( ) const
```

Gets the filter name.

8.171 rti::core::LocatorFilterElement Class Reference

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Specifies the configuration of an individual channel within a MultiChannel DataWriter.

```
#include <LocatorFilter.hpp>
```

Public Member Functions

- **LocatorFilterElement** (const std::string &the_filter_expression, const rti::core::LocatorSeq &the_locators)

  Creates an instance with the provided filter_expression and locators.
- rti::core::LocatorSeq **locators** () const

  Get the locators associated with this filter.
- **LocatorFilterElement** & **locators** (const LocatorSeq &the_locators)

  Set the locators associated with this LocatorFilterImpl.
- std::string **filter_expression** () const

  Get the filter expression.
- **LocatorFilterElement** & **filter_expression** (const std::string &the_expression)

  Set the filter expression.

8.171.1 Detailed Description

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Specifies the configuration of an individual channel within a MultiChannel DataWriter.

See also

dds::core::policy::LocatorFilter

8.171.2 Constructor & Destructor Documentation

8.171.2.1 LocatorFilterElement()

```
rti::core::LocatorFilterElement::LocatorFilterElement (
            const std::string & the_filter_expression,
            const rti::core::LocatorSeq & the_locators )  [inline]
```

Creates an instance with the provided filter_expression and locators.

Parameters

the_filter_expression	A logical expression used to determine the data that will be published in the channel.
the_locators	Sequence containing from one to four **rti::core::Locator** (p. 1673)

8.171.3 Member Function Documentation

8.171.3.1 locators() [1/2]

```
rti::core::LocatorSeq rti::core::LocatorFilterElement::locators ( ) const  [inline]
```

Get the locators associated with this filter.

The locators are a sequence containing from one to four **rti::core::Locator** (p. 1673), used to specify the multicast address locators of an individual channel within a MultiChannel DataWriter.

8.171.3.2 locators() [2/2]

```
LocatorFilterElement & rti::core::LocatorFilterElement::locators (
            const LocatorSeq & the_locators )  [inline]
```

Set the locators associated with this LocatorFilterImpl.

8.171.3.3 filter_expression() [1/2]

```
std::string rti::core::LocatorFilterElement::filter_expression ( ) const  [inline]
```

Get the filter expression.

8.171.3.4 filter_expression() [2/2]

```
LocatorFilterElement & rti::core::LocatorFilterElement::filter_expression (
            const std::string & the_expression )  [inline]
```

Set the filter expression.

The filter expression is a logical expression used to determine the data that will be published in the channel.

If the expression evaluates to TRUE, a sample will be published on the channel.

An empty string always evaluates the expression to TRUE.

The syntax of the expression will depend on the value of **rti::core::policy::LocatorFilter::filter_name** (p. 1678)

See also

Queries and Filters Syntax (p. 84)

[default] NULL (invalid value)

8.172 rti::core::LocatorKind_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) LocatorKind.

`#include <Locator.hpp>`

Public Types

- enum **type** {
  INVALID = NDDS_TRANSPORT_CLASSID_INVALID ,
  ANY = NDDS_TRANSPORT_CLASSID_ANY ,
  UDPv4 = NDDS_TRANSPORT_CLASSID_UDPv4 ,
  UDPv4_WAN = NDDS_TRANSPORT_CLASSID_UDPv4_WAN ,
  SHMEM = NDDS_TRANSPORT_CLASSID_SHMEM ,
  SHMEM_510 = NDDS_TRANSPORT_CLASSID_SHMEM_510 ,
  INTRA = NDDS_TRANSPORT_CLASSID_INTRA ,
  UDPv6 = NDDS_TRANSPORT_CLASSID_UDPv6 ,
  UDPv6_510 = NDDS_TRANSPORT_CLASSID_UDPv6_510 ,
  TCPV4_LAN = NDDS_TRANSPORT_CLASSID_TCPV4_LAN ,
  TCPV4_WAN = NDDS_TRANSPORT_CLASSID_TCPV4_WAN ,
  TLSV4_LAN = NDDS_TRANSPORT_CLASSID_TLSV4_LAN ,
  TLSV4_WAN = NDDS_TRANSPORT_CLASSID_TLSV4_WAN ,
  RESERVED = NDDS_TRANSPORT_CLASSID_RESERVED_RANGE }

  The underlying `enum` type.

8.172.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) LocatorKind.

8.172.2 Member Enumeration Documentation

8.172.2.1 type

`enum` **rti::core::LocatorKind_def::type**

The underlying `enum` type.

Enumerator

INVALID	An invalid locator.
UDPv4	A locator for a UDPv4 address.
UDPv4_WAN	A locator for a UDPv4 asymmetric transport address.
SHMEM	A locator for an address acessed via shared memory.
UDPv6	A locator for a UDPv6 address.

8.173 rti::config::LogCategory_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) LogCategory.

`#include <rti/config/Logger.hpp>`

Public Types

- enum **type** {
  platform ,
  communication ,
  database ,
  entities ,
  api ,
  discovery ,
  security ,
  user ,
  all_categories ,
  PLATFORM = NDDS_CONFIG_LOG_CATEGORY_PLATFORM ,
  COMMUNICATION = NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION ,
  DATABASE = NDDS_CONFIG_LOG_CATEGORY_DATABASE ,
  ENTITIES = NDDS_CONFIG_LOG_CATEGORY_ENTITIES ,
  API = NDDS_CONFIG_LOG_CATEGORY_API ,
  DISCOVERY = NDDS_CONFIG_LOG_CATEGORY_DISCOVERY ,
  SECURITY = NDDS_CONFIG_LOG_CATEGORY_SECURITY ,
  ALL_CATEGORIES = NDDS_CONFIG_LOG_CATEGORY_ALL }

  The underlying `enum` type.

8.173.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) LogCategory.

8.173.2 Member Enumeration Documentation

8.173.2.1 type

`enum` **rti::config::LogCategory_def::type**

The underlying `enum` type.

Enumerator

platform	Log messages pertaining to the underlying platform (hardware and OS) on which RTI Connext is running are in this category.

Enumerator

communication	Log messages pertaining to data serialization and deserialization and network traffic are in this category.
database	Log messages pertaining to the internal database in which RTI Connext objects are stored are in this category.
entities	Log messages pertaining to local and remote entities, and to a subset of the discovery process, are in this category. (To see all discovery-related messages, use the DISCOVERY category.)
api	Log messages pertaining to the API layer of RTI Connext (such as method argument validation) are in this category.
discovery	Log messages pertaining to discovery are in this category.
security	Log messages pertaining to security are in this category. These messages include any messages logged by RTI Connext components even if they are not related to the Security Plugins.
user	Log messages that are generated by the user using the following log APIs: **rti::config::Logger::emergency** (p. 1689), **rti::config::Logger::alert** (p. 1689), **rti::config::Logger::critical** (p. 1689), **rti::config::Logger::error** (p. 1689), **rti::config::Logger::warning** (p. 1690), **rti::config::Logger::notice** (p. 1690), **rti::config::Logger::informational** (p. 1690), **rti::config::Logger::debug** (p. 1690).
all_categories	Log messages pertaining to all categories in RTI Connext.

8.174 rti::config::Logger Class Reference

The singleton type used to configure RTI Connext logging.

```
#include <rti/config/Logger.hpp>
```

Public Member Functions

- **Verbosity verbosity** ()

  Get the verbosity at which RTI Connext is currently logging diagnostic information.
- **Verbosity verbosity_by_category** (**LogCategory** category)

  Get the verbosity at which RTI Connext is currently logging diagnostic information in the given category.
- void **verbosity** (**Verbosity** verbosity)

  Set the verbosity at which RTI Connext will log diagnostic information.
- void **verbosity_by_category** (**LogCategory** category, **Verbosity verbosity**)

  Set the verbosity at which RTI Connext will log diagnostic information in the given category.
- FILE * **output_file** ()

  Get the file to which the logged output is redirected.
- void **output_file** (FILE *out)

  Set the file to which the logged output is redirected.
- void **output_file** (const char *file_name)

  Set the name of the file to which the logged output is redirected.
- void **output_file_set** (const char *file_preffix, const char *file_suffix, int max_bytes, int max_files= **dds::core**↩ **::LENGTH_UNLIMITED**)

Configure a set of files to redirect the logged output.

- void **output_handler** (std::function< void(const **LogMessage** &)> handler)

  Redirects RTI Connext logging to a function.

- void **reset_output_handler** ()

  Resets a previously set output handler.

- **PrintFormat print_format** ()

  *Get the current message format for the log level **rti::config::LogLevel_def::EXCEPTION** (p. 1693).*

- void **print_format** (**PrintFormat** print_format)

  *Set the message format that RTI Connext will use to log diagnostic information for all the log levels, except for **rti::config::LogLevel_def::FATAL_ERROR** (p. 1693). When the **Activity Context** (p. 449) is printed, the user can select the information that will be part of the **Activity Context** (p. 449) by using the API **rti::config::activity_context::set_attribute_mask()** (p. 450).*

- **PrintFormat print_format_by_log_level** (**LogLevel** log_level)

  Get the current message format, by log level, that RTI Connext is using to log diagnostic information.

- void **print_format_by_log_level** (**PrintFormat print_format**, **LogLevel** log_level)

  *Set the message format, by log level, that RTI Connext will use to log diagnostic information. When the **Activity Context** (p. 449) is printed, the user can select the information that will be part of the **Activity Context** (p. 449) by using the API **rti::config::activity_context::set_attribute_mask()** (p. 450).*

- void **emergency** (const char *msg)

  *Logs message with **rti::config::SyslogLevel::emergency** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **alert** (const char *msg)

  *Logs message with **rti::config::SyslogLevel::alert** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **critical** (const char *msg)

  *Logs message with **rti::config::SyslogLevel::critical** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **error** (const char *msg)

  *Logs message with **rti::config::SyslogLevel::error** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **warning** (const char *msg)

  *Logs message with **rti::config::SyslogLevel::warning** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **notice** (const char *msg)

  *Logs message with **rti::config::SyslogLevel::notice** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **informational** (const char *msg)

  *Logs message with **rti::config::SyslogLevel::informational** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **debug** (const char *msg)

  *Logs message with **rti::config::SyslogLevel::debug** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **emergency** (std::string_view msg)

  *Logs message with **rti::config::SyslogLevel::emergency** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **alert** (std::string_view msg)

  *Logs message with **rti::config::SyslogLevel::alert** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **critical** (std::string_view msg)

  *Logs message with **rti::config::SyslogLevel::critical** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **error** (std::string_view msg)

  *Logs message with **rti::config::SyslogLevel::error** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **warning** (std::string_view msg)

  *Logs message with **rti::config::SyslogLevel::warning** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **notice** (std::string_view msg)

  *Logs message with **rti::config::SyslogLevel::notice** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..*

- void **informational** (std::string_view msg)

Logs message with ***rti::config::SyslogLevel::informational*** *(p. 458) and* ***rti::config::LogCategory_def::user*** *(p. 1683)..*

- void **debug** (std::string_view msg)

  Logs message with ***rti::config::SyslogLevel::debug*** *(p. 458) and* ***rti::config::LogCategory_def::user*** *(p. 1683)..*

Static Public Member Functions

- static **Logger** & **instance** ()

  Get the singleton instance of this type.

8.174.1 Detailed Description

The singleton type used to configure RTI Connext logging.

8.174.2 Member Function Documentation

8.174.2.1 instance()

```
static Logger & rti::config::Logger::instance ( ) [inline], [static]
```

Get the singleton instance of this type.

Examples

Foo_publisher.cxx, and **Foo_subscriber.cxx**.

Referenced by **rti::config::ScopedLoggerVerbosity::ScopedLoggerVerbosity()**, and **rti::config::Scoped↩ LoggerVerbosity::∼ScopedLoggerVerbosity()**.

8.174.2.2 verbosity() [1/2]

```
Verbosity rti::config::Logger::verbosity ( )
```

Get the verbosity at which RTI Connext is currently logging diagnostic information.

The default verbosity if **rti::config::Logger::verbosity** (p. 1685) is never called is **rti::config::Verbosity_def↩ ::exception** (p. 2614).

If **rti::config::Logger::verbosity_by_category** (p. 1685) has been used to set different verbosities for different categories of messages, this method will return the maximum verbosity of all categories.

Examples

Foo_publisher.cxx, and **Foo_subscriber.cxx**.

Referenced by **rti::config::ScopedLoggerVerbosity::ScopedLoggerVerbosity()**, and **rti::config::Scoped↩ LoggerVerbosity::∼ScopedLoggerVerbosity()**.

8.174.2.3 verbosity_by_category() [1/2]

```
Verbosity rti::config::Logger::verbosity_by_category (
            LogCategory category )
```

Get the verbosity at which RTI Connext is currently logging diagnostic information in the given category.

The default verbosity if **rti::config::Logger::verbosity** (p. 1685) and **rti::config::Logger::verbosity_by_category** (p. 1685) are never called is **rti::config::Verbosity_def::exception** (p. 2614).

8.174.2.4 verbosity() [2/2]

```
void rti::config::Logger::verbosity (
            Verbosity verbosity )
```

Set the verbosity at which RTI Connext will log diagnostic information.

Note: Logging at high verbosities will be detrimental to your application's performance. Your default setting should typically remain at **rti::config::Verbosity_def::warning** (p. 2614) or below. (The default verbosity if you never set it is **rti::config::Verbosity_def::exception** (p. 2614).)

8.174.2.5 verbosity_by_category() [2/2]

```
void rti::config::Logger::verbosity_by_category (
            LogCategory category,
            Verbosity verbosity )
```

Set the verbosity at which RTI Connext will log diagnostic information in the given category.

8.174.2.6 output_file() [1/3]

```
FILE * rti::config::Logger::output_file ( )
```

Get the file to which the logged output is redirected.

If no output file has been registered through **rti::config::Logger::output_file** (p. 1686), this method will return NULL. In this case, logged output will on most platforms go to standard out as if through printf.

8.174.2.7 output_file() [2/3]

```
void rti::config::Logger::output_file (
            FILE * out )
```

Set the file to which the logged output is redirected.

The file passed may be NULL, in which case further logged output will be redirected to the platform-specific default output location (standard out on most platforms).

For better performance when log messages are generated frequently, the log messages are not flushed into a file immediately after they are generated. In other words, while writing a log message, RTI Connext only calls the function fwrite() (see https://pubs.opengroup.org/onlinepubs/009695399/functions/fwrite.html); it does not call the function fflush() (see https://pubs.opengroup.org/onlinepubs/009695399/functions/fflush.html). If your application requires a different flushing behavior, you may use **rti::config::Logger::output_handler** (p. 1687) to configure a custom logging device.

8.174.2.8 output_file() [3/3]

```
void rti::config::Logger::output_file (
            const char * file_name )
```

Set the name of the file to which the logged output is redirected.

The name may be NULL, in which case further logged output will be redirected to the platform-specific default output location (standard out on most platforms).

See **rti::config::Logger::output_file** (p. 1686) for the flushing behavior.

8.174.2.9 output_file_set()

```
void rti::config::Logger::output_file_set (
            const char * file_preffix,
            const char * file_suffix,
            int max_bytes,
            int max_files = dds::core::LENGTH_UNLIMITED )
```

Configure a set of files to redirect the logged output.

The logged output will be redirected to a set of files whose names are configured with a prefix and a suffix. The maximum number of bytes configures how many bytes to write into a file before opening the next file. After reaching the maximum number of files, the first one is overwritten.

For example, if the prefix is '**Foo** (p. 1588)', the suffix is '.txt', the max number of bytes is 1GB, and the max number of files is 3, the logger will create (at most) these files: Foo1.txt, Foo2.txt, and Foo3.txt. It will write to Foo1.txt, and after writing 1GB, it will move on to Foo2.txt, then to Foo3.txt, then to Foo1.txt again, and so on.

To stop logging to these files and redirect the output to the platform-specific location, pass NULL, NULL, 0, 0.

See **rti::config::Logger::output_file** (p. 1686) for the flushing behavior.

8.174.2.10 output_handler()

```
void rti::config::Logger::output_handler (
            std::function< void(const LogMessage &)> handler ) [inline]
```

Redirects RTI Connext logging to a function.

When a handler is set, log messages stop being directed to the standard output and are passed to the handler.

To direct logging back to the standard output, use **reset_output_handler()** (p. 1688).

Parameters

handler	A function or function object capable of receiving a `const` **LogMessage** (p. 1693) `&` argument, the log message.

The following example sets a handler that saves all log messages in a vector:

```
std::vector<std::string> saved_logs;
Logger::instance().output_handler([&saved_logs](const LogMessage& message) {
    saved_logs.push_back(message.text); // message is copied as a std::string
});
```

The `char *` in the **LogMessage** (p. 1693) is only valid within the callback.

8.174.2.11 reset_output_handler()

```
void rti::config::Logger::reset_output_handler ( )  [inline]
```

Resets a previously set output handler.

Directs RTI Connext logging back to the standard output.

8.174.2.12 print_format() [1/2]

```
PrintFormat rti::config::Logger::print_format ( )
```

Get the current message format for the log level **rti::config::LogLevel_def::EXCEPTION** (p. 1693).

Use rti::config::Logger::get_print_format_by_log_level to retrieve the format for other log levels.

If **rti::config::Logger::print_format** (p. 1688) is never called, the default format is **PrintFormat_def::DEFAULT** (p. 1961).

8.174.2.13 print_format() [2/2]

```
void rti::config::Logger::print_format (
              PrintFormat print_format )
```

Set the message format that RTI Connext will use to log diagnostic information for all the log levels, except for **rti::config::LogLevel_def::FATAL_ERROR** (p. 1693). When the **Activity Context** (p. 449) is printed, the user can select the information that will be part of the **Activity Context** (p. 449) by using the API **rti::config::activity_context::set_attribute_mask()** (p. 450).

8.174.2.14 print_format_by_log_level() [1/2]

```
PrintFormat rti::config::Logger::print_format_by_log_level (
            LogLevel log_level )
```

Get the current message format, by log level, that RTI Connext is using to log diagnostic information.

If **rti::config::Logger::print_format** (p. 1688) is never called, the default format is **PrintFormat_def::DEFAULT** (p. 1961).

8.174.2.15 print_format_by_log_level() [2/2]

```
void rti::config::Logger::print_format_by_log_level (
            PrintFormat print_format,
            LogLevel log_level )
```

Set the message format, by log level, that RTI Connext will use to log diagnostic information. When the **Activity Context** (p. 449) is printed, the user can select the information that will be part of the **Activity Context** (p. 449) by using the API **rti::config::activity_context::set_attribute_mask()** (p. 450).

8.174.2.16 emergency() [1/2]

```
void rti::config::Logger::emergency (
            const char * msg )
```

Logs message with **rti::config::SyslogLevel::emergency** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.17 alert() [1/2]

```
void rti::config::Logger::alert (
            const char * msg )
```

Logs message with **rti::config::SyslogLevel::alert** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.18 critical() [1/2]

```
void rti::config::Logger::critical (
            const char * msg )
```

Logs message with **rti::config::SyslogLevel::critical** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.19 error() [1/2]

```
void rti::config::Logger::error (
            const char * msg )
```

Logs message with **rti::config::SyslogLevel::error** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.20 warning() [1/2]

```
void rti::config::Logger::warning (
            const char * msg )
```

Logs message with **rti::config::SyslogLevel::warning** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.21 notice() [1/2]

```
void rti::config::Logger::notice (
            const char * msg )
```

Logs message with **rti::config::SyslogLevel::notice** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.22 informational() [1/2]

```
void rti::config::Logger::informational (
            const char * msg )
```

Logs message with **rti::config::SyslogLevel::informational** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.23 debug() [1/2]

```
void rti::config::Logger::debug (
            const char * msg )
```

Logs message with **rti::config::SyslogLevel::debug** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.24 emergency() [2/2]

```
void rti::config::Logger::emergency (
            std::string_view msg )
```

Logs message with **rti::config::SyslogLevel::emergency** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.25 alert() [2/2]

```
void rti::config::Logger::alert (
            std::string_view msg )
```

Logs message with **rti::config::SyslogLevel::alert** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.26 critical() [2/2]

```
void rti::config::Logger::critical (
            std::string_view msg )
```

Logs message with **rti::config::SyslogLevel::critical** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.27 error() [2/2]

```
void rti::config::Logger::error (
            std::string_view msg )
```

Logs message with **rti::config::SyslogLevel::error** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.28 warning() [2/2]

```
void rti::config::Logger::warning (
            std::string_view msg )
```

Logs message with **rti::config::SyslogLevel::warning** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.29 notice() [2/2]

```
void rti::config::Logger::notice (
            std::string_view msg )
```

Logs message with **rti::config::SyslogLevel::notice** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.30 informational() [2/2]

```
void rti::config::Logger::informational (
            std::string_view msg )
```

Logs message with **rti::config::SyslogLevel::informational** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.174.2.31 debug() [2/2]

```
void rti::config::Logger::debug (
            std::string_view msg )
```

Logs message with **rti::config::SyslogLevel::debug** (p. 458) and **rti::config::LogCategory_def::user** (p. 1683)..

8.175 rti::config::LogLevel_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) LogLevel;.

```
#include <rti/config/Logger.hpp>
```

Public Types

- enum **type** {
  FATAL_ERROR ,
  EXCEPTION ,
  WARNING ,
  STATUS_LOCAL ,
  STATUS_REMOTE ,
  STATUS_ALL }

  The underlying `enum` *type.*

8.175.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) LogLevel;.

8.175.2 Member Enumeration Documentation

8.175.2.1 type

enum **rti::config::LogLevel_def::type**

The underlying enum type.

Enumerator

FATAL_ERROR	The message describes a fatal error. A fatal error indicates an unrecoverable situation in the functioning of RTI Connext. Error messages with this log level usually indicate a violation of an internal invariant or a segfault, and may include the function call stack where the fatal error happened.
EXCEPTION	The message describes an error. An error indicates a non-fatal problem in the functioning of RTI Connext. Errors are usually recoverable and will not stop application execution, although they may prevent some features from working properly. The most common cause of non-fatal errors is incorrect configuration and incorrect arguments.
WARNING	The message describes a warning. A warning indicates that RTI Connext is taking an action that may or may not be what you intended. Some configuration information is also logged at this verbosity to aid in debugging.
STATUS_LOCAL	The message contains information about the lifecycles of local RTI Connext objects will be logged.
STATUS_REMOTE	The message contains information about the lifecycles of remote RTI Connext objects will be logged.
STATUS_ALL	The message contains debug information that might be relevant to your application.

8.176 rti::config::LogMessage Struct Reference

A log message's input parameter to an output handler, including the text and additional information.

#include <rti/config/Logger.hpp>

Public Attributes

- **LogLevel level**

The log level of this message.

- const char * **text**

  The text of this message.

- bool **is_security_message**

  Indicates if the message is a security-related message.

- bool **is_license_message**

  Indicates if the message is a license-related message.

- unsigned int **message_id**

  A numeric code that identifies a specific log message.

- **dds::core::Duration timestamp**

  The time when the log message was printed.

- **LogFacility facility**

  The Facility associated with the log message.

8.176.1 Detailed Description

A log message's input parameter to an output handler, including the text and additional information.

This struct passed to the handler set in **Logger::output_handler()** (p. 1687).

8.176.2 Member Data Documentation

8.176.2.1 level

```
LogLevel rti::config::LogMessage::level
```

The log level of this message.

8.176.2.2 text

```
const char* rti::config::LogMessage::text
```

The text of this message.

8.176.2.3 is_security_message

```
bool rti::config::LogMessage::is_security_message
```

Indicates if the message is a security-related message.

A "security-related message" is a log message that meets any of the following:

- A security event logged with the RTI Security Plugins Logging Plugin.
- Any regular message coming from the RTI Security Plugins.
- RTI TLS Support log messages related to OpenSSL (e.g., SSL handshake failures or certificate validation failures).

8.176.2.4 is_license_message

```
bool rti::config::LogMessage::is_license_message
```

Indicates if the message is a license-related message.

When using a licensed version of RTI Connext (such as the RTI Connext Express edition), certain messages related to the license, such as the trial expiration date or license limitations, are logged when a DomainParticipant is created. This flag indicates whether the message is one of those license-related messages.

8.176.2.5 message_id

```
unsigned int rti::config::LogMessage::message_id
```

A numeric code that identifies a specific log message.

8.176.2.6 timestamp

```
dds::core::Duration rti::config::LogMessage::timestamp
```

The time when the log message was printed.

8.176.2.7 facility

```
LogFacility rti::config::LogMessage::facility
```

The Facility associated with the log message.

See also

LogFacility (p. 459)

8.177 rti::core::LongDouble Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Encapsulates an IDL `long double`

`#include <LongDouble.hpp>`

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **LongDouble** ()
  Creates a long double with value 0.
- char **operator[ ]** (size_t i) const
  Access a byte (0 to 15)
- char & **operator[ ]** (size_t i)
  Access a byte (0 to 15)

8.177.1 Detailed Description

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Encapsulates an IDL `long double`

Since the representation of a C++ `long double` is not portable, this class encapsulates an **IDL** (p. 606) 16-byte `long double`.

8.177.2 Constructor & Destructor Documentation

8.177.2.1 LongDouble()

```
rti::core::LongDouble::LongDouble ( )  [inline]
```

Creates a long double with value 0.

8.177.3 Member Function Documentation

8.177.3.1 operator[]() [1/2]

```
char rti::core::LongDouble::operator[] (
            size_t i ) const  [inline]
```

Access a byte (0 to 15)

8.177.3.2 operator[]() [2/2]

```
char & rti::core::LongDouble::operator[] (
            size_t i )  [inline]
```

Access a byte (0 to 15)

8.178 dds::sub::DataReader< T >::ManipulatorSelector Class Reference

A **Selector** (p. 2320) class enabling the streaming API.

`#include <TDataReader.hpp>`

Public Member Functions

- **ManipulatorSelector** & **operator**>> (**dds::sub::LoanedSamples**< T > &samples)

  *Streaming operator taking a **LoanedSamples** (p. 1663) object.*
- **ManipulatorSelector** & **operator**>> (bool(*manipulator)(**ReadModeDummyType**))

  Streaming operator taking in a stream manipulator that will determine whether the samples will be read or taken.
- template<typename Functor >
  ManipulatorSelector & **operator**>> (Functor f)

  Streaming operator taking in a Functor.

8.178.1 Detailed Description

template<typename T>
class dds::sub::DataReader< T >::ManipulatorSelector

A **Selector** (p. 2320) class enabling the streaming API.

Similar to the **Selector** (p. 2320) class, the **ManipulatorSelector** (p. 1697) class is used by the **DataReader** (p. 976) to compose read and take operations using the streaming operator >>.

A **ManipulatorSelector** (p. 1697) has an associated **DataReader** (p. 976) and configures the behavior of the read or take operation performed by that **DataReader** (p. 976).

The **ManipulatorSelector** (p. 1697) works by using a number of functions which configure the stream:

- **read(dds::sub::ReadModeDummyType)** (p. 1024)
- **take(dds::sub::ReadModeDummyType)** (p. 1024)
- **dds::sub::max_samples(uint32_t n)** (p. 662)
- **dds::sub::content(const dds::sub::Query& query)** (p. 663)
- **dds::sub::condition(const dds::sub::cond::ReadCondition& condition)** (p. 663)
- **dds::sub::state(const dds::sub::status::DataState& s)** (p. 664)
- **dds::sub::instance(const dds::core::InstanceHandle& h)** (p. 665)
- **dds::sub::next_instance(const dds::core::InstanceHandle& h)** (p. 665)

The above functions all return functors. You do not and should not use these functors directly:

- dds::sub::functors::MaxSamplesManipulatorFunctor
- dds::sub::functors::ContentFilterManipulatorFunctor
- dds::sub::functors::ConditionManipulatorFunctor
- dds::sub::functors::StateFilterManipulatorFunctor
- dds::sub::functors::InstanceManipulatorFunctor
- dds::sub::functors::NextInstanceManipulatorFunctor

For example, to perform a read of at most 5 unread samples:

```
LoanedSamples<Foo> samples;
reader » read
       » max_samples(5)
       » state(dds::sub::status::DataState::new_data())
       » samples;
```

See also

Accessing Received Data (p. 318)

8.178.2 Member Function Documentation

8.178.2.1 operator>>() [1/3]

```
template<typename T >
ManipulatorSelector &  dds::sub::DataReader< T >::ManipulatorSelector>> (
              dds::sub::LoanedSamples< T > & samples )  [inline]
```

Streaming operator taking a **LoanedSamples** (p. 1663) object.

This operator allows you to direct the outcome of a string of stream operators into a **LoanedSamples** (p. 1663) object.

Parameters

samples	The **LoanedSamples** (p. 1663) container to fill with the read/taken samples.

8.178.2.2 operator>>() [2/3]

```
template<typename T >
ManipulatorSelector &  dds::sub::DataReader< T >::ManipulatorSelector>> (
             bool(*)( ReadModeDummyType) manipulator )  [inline]
```

Streaming operator taking in a stream manipulator that will determine whether the samples will be read or taken.

The provided manipulator for this operator will be either **dds::sub::read** (p. 661) or **dds::sub::take** (p. 662).

Parameters

manipulator	Either **dds::sub::read** (p. 661) or **dds::sub::take** (p. 662)

See also

dds::sub::read(dds::sub::ReadModeDummyType) (p. 661)

dds::sub::take(dds::sub::ReadModeDummyType) (p. 662)

8.178.2.3 operator>>() [3/3]

```
template<typename T >
template<typename Functor >
```

```
ManipulatorSelector & dds::sub::DataReader< T >::ManipulatorSelector::operator>> (
            Functor f ) [inline]
```

Streaming operator taking in a Functor.

There are a number of functions which return the Functors that this operator expects:

- **dds::sub::max_samples(uint32_t n)** (p. 662)
- **dds::sub::content(const dds::sub::Query& query)** (p. 663)
- **dds::sub::condition(const dds::sub::cond::ReadCondition& condition)** (p. 663)
- **dds::sub::state(const dds::sub::status::DataState& s)** (p. 664)
- **dds::sub::instance(const dds::core::InstanceHandle& h)** (p. 665)
- **dds::sub::next_instance(const dds::core::InstanceHandle& h)** (p. 665)

Parameters

f	A Functor that is returned from one of the above functions

Returns

ManipulatorSelector (p. 1697)&

8.179 rti::sub::ManipulatorSelector< T > Class Template Reference

```
#include <SelectorImpl.hpp>
```

Inherits rti::sub::DefaultSelector< T >.

8.179.1 Detailed Description

template<typename T>
class rti::sub::ManipulatorSelector< T >

Selector class enabling the streaming API.

8.180 dds::core::xtypes::Member Class Reference

<<***value-type***>> *(p. 350)* Represents a **StructType** (p. 2400) member

```
#include "dds/core/xtypes/MemberType.hpp"
```

Public Member Functions

- **Member** (const std::string & **name**, const DynamicTypeImpl & **type**)

  Creates a member consisting of a name and a type.
- **Member** (const std::string &the_name, DynamicTypeImpl &&the_type)

  *<<**C++11**>> (p. 353) Creates a member consisting of a name and a type.*
- const **dds::core::string** & **name** () const

  Gets the member name.
- **dds::core::string** & **name** ()

  Gets the member name.
- const **DynamicType** & **type** ()

  Gets the member type.
- bool **has_id** () const

  Indicates it the member has an ID annotation.
- int32_t **get_id** () const

  Returns the ID annotation of this member.
- bool **is_pointer** () const

  *<<**extension**>> (p. 354) Indicates if this member is a pointer*
- bool **is_key** () const

  Checks if a member is a key.
- bool **is_optional** () const

  Checks if a member is optional.
- bool **is_must_understand** () const

  Checks if a member is marked as "must understand".
- bool **is_bitset** () const

  Checks if a member is a bitset.
- bool **has_bitbound** () const

  Unsupported, always returns false.
- int32_t **get_bitbound** () const

  Unsupported, always returns 32.
- **Member** & **name** (const **dds::core::string** &value)

  Sets the member name.
- **Member** & **key** (bool value)

  Sets the key annotation of a member.
- **Member** & **optional** (bool value)

  Sets the optional annotation of a member.
- **Member** & **must_understand** (bool value)

  Sets the "must understand" annotation of a member.
- **Member** & **id** (int32_t value)

  Sets the ID annotation of a member.
- **Member** & **pointer** (bool value)

  *<<**extension**>> (p. 354) Makes a member a pointer*

Static Public Attributes

- static const int32_t **INVALID_ID**

  The special ID of a member without the ID annotation.

8.180.1 Detailed Description

<<***value-type***>> *(*p. *350)* Represents a **StructType** (p. 2400) member

Encapsulates the name and type of a **StructType** (p. 2400) member along with several IDL annotations (key, optional, bitset, bitbound, id).

8.180.2 Constructor & Destructor Documentation

8.180.2.1 Member() [1/2]

```
dds::core::xtypes::Member::Member (
            const std::string & name,
            const DynamicTypeImpl & type )
```

Creates a member consisting of a name and a type.

8.180.2.2 Member() [2/2]

```
dds::core::xtypes::Member::Member (
            const std::string & the_name,
            DynamicTypeImpl && the_type )  [inline]
```

<<***C++11***>> *(*p. *353)* Creates a member consisting of a name and a type.

The type is moved.

8.180.3 Member Function Documentation

8.180.3.1 name() [1/3]

```
const  dds::core::string & dds::core::xtypes::Member::name ( ) const
```

Gets the member name.

8.180.3.2 name() [2/3]

```
dds::core::string & dds::core::xtypes::Member::name ( )
```

Gets the member name.

8.180.3.3 type()

```
const  DynamicType & dds::core::xtypes::Member::type ( )
```

Gets the member type.

8.180.3.4 has_id()

```
bool dds::core::xtypes::Member::has_id ( ) const
```

Indicates it the member has an ID annotation.

True if this member has been annotated explicitly with an ID. False if the has a default-assigned ID.

8.180.3.5 get_id()

```
int32_t dds::core::xtypes::Member::get_id ( ) const
```

Returns the ID annotation of this member.

If **has_id()** (p. 1703) is true, this returns the value of the ID annotation. Otherwise it returns INVALID_ID.

8.180.3.6 is_pointer()

```
bool dds::core::xtypes::Member::is_pointer ( ) const
```

<<***extension***>> *(p. 354)* Indicates if this member is a pointer

8.180.3.7 is_key()

```
bool dds::core::xtypes::Member::is_key ( ) const
```

Checks if a member is a key.

This corresponds to the @Key IDL member annotation

8.180.3.8 is_optional()

```
bool dds::core::xtypes::Member::is_optional ( ) const
```

Checks if a member is optional.

This corresponds to the @Optional IDL member annotation

8.180.3.9 is_must_understand()

```
bool dds::core::xtypes::Member::is_must_understand ( ) const
```

Checks if a member is marked as "must understand".

This corresponds to the @must_understand IDL member annotation.

8.180.3.10 is_bitset()

```
bool dds::core::xtypes::Member::is_bitset ( ) const
```

Checks if a member is a bitset.

8.180.3.11 has_bitbound()

```
bool dds::core::xtypes::Member::has_bitbound ( ) const
```

Unsupported, always returns false.

8.180.3.12 get_bitbound()

```
int32_t dds::core::xtypes::Member::get_bitbound ( ) const
```

Unsupported, always returns 32.

8.180.3.13 name() [3/3]

```
Member & dds::core::xtypes::Member::name (
            const  dds::core::string & value )
```

Sets the member name.

8.180.3.14 key()

```
Member & dds::core::xtypes::Member::key (
            bool value )
```

Sets the key annotation of a member.

Postcondition

If **optional()** (p. 1705) was true and key is set to true, **optional()** (p. 1705) becomes false, because a member can't be both a key and optional.

[default] false

8.180.3.15 optional()

```
Member & dds::core::xtypes::Member::optional (
            bool value )
```

Sets the optional annotation of a member.

Postcondition

If **key()** (p. 1704) was true and optional is set to true, **key()** (p. 1704) becomes false, because a member can't be both a key and optional.

[default] false

8.180.3.16 must_understand()

```
Member & dds::core::xtypes::Member::must_understand (
            bool value )
```

Sets the "must understand" annotation of a member.

[default] false

8.180.3.17 id()

```
Member & dds::core::xtypes::Member::id (
            int32_t value )
```

Sets the ID annotation of a member.

[default] Automatically assigned as the ID of the previous member plus one previous member

8.180.3.18 pointer()

```
Member & dds::core::xtypes::Member::pointer (
            bool value )
```

<<***extension***>> *(*p. *354)* Makes a member a pointer

[default] false

8.180.4 Member Data Documentation

8.180.4.1 INVALID_ID

```
const int32_t dds::core::xtypes::Member::INVALID_ID  [static]
```

The special ID of a member without the ID annotation.

8.181 rti::core::policy::Monitoring Class Reference

<<***extension***>> *(*p. *116)* Configures the use of the RTI **Monitoring** (p. 1706) Library 2.0 to collect and distribute RTI Connext telemetry data.

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **Monitoring** ()

  Creates the default policy (monitoring is disabled).
- **Monitoring** & **enable** (bool the_enable)

  <<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116) Enables the collection and distribution of telemetry data for an RTI Connext application using RTI **Monitoring** (*p. *1706) Library 2.0.*
- bool **enable** () const

  Getter (see setter with the same name).
- **Monitoring** & **application_name** (const **rti::core::optional_value**< std::string > &the_application_name)

  Sets the name of the resource that represents this RTI Connext application.
- **Monitoring** & **application_name** (const char *the_application_name)

  Sets the name of the resource that represents this RTI Connext application.
- **rti::core::optional_value**< std::string > **application_name** () const

  Getter (see setter with the same name).
- **Monitoring** & **distribution_settings** (const **MonitoringDistributionSettings** &the_distribution_settings)

  <<***extension***>> *(*p. *116) Configures the distribution of telemetry data.*
- const **MonitoringDistributionSettings** & **distribution_settings** () const

Gets the distribution settings by const-reference (see setter).

- **MonitoringDistributionSettings** & **distribution_settings** ()

  Gets the distribution settings by reference (see setter).

- **Monitoring** & **telemetry_data** (const **MonitoringTelemetryData** &the_telemetry_data)

  *<<**extension**>> (p. 116) Configures the telemetry data that will be distributed.*

- const **MonitoringTelemetryData** & **telemetry_data** () const

  Gets the telemetry data by const-reference (see setter).

- **MonitoringTelemetryData** & **telemetry_data** ()

  Gets the telemetry data by reference (see setter).

Static Public Member Functions

- static **Monitoring** **Enabled** ()

  *Returns an instance that enables **Monitoring** (p. 1706) with default settings.*

- static **Monitoring** **Disabled** ()

  *Returns an instance that disables **Monitoring** (p. 1706).*

8.181.1 Detailed Description

*<<**extension**>> (p. 116)* Configures the use of the RTI **Monitoring** (p. 1706) Library 2.0 to collect and distribute RTI Connext telemetry data.

8.181.2 Constructor & Destructor Documentation

8.181.2.1 Monitoring()

```
rti::core::policy::Monitoring::Monitoring ( )  [inline]
```

Creates the default policy (monitoring is disabled).

8.181.3 Member Function Documentation

8.181.3.1 Enabled()

```
static  Monitoring rti::core::policy::Monitoring::Enabled ( )  [inline], [static]
```

Returns an instance that enables **Monitoring** (p. 1706) with default settings.

8.181.3.2 Disabled()

```
static  Monitoring rti::core::policy::Monitoring::Disabled ( )  [inline], [static]
```

Returns an instance that disables **Monitoring** (p. 1706).

8.181.3.3 enable() [1/2]

```
Monitoring & rti::core::policy::Monitoring::enable (
            bool the_enable )
```

*<<**basic**>> (p. 104) <<**extension**>> (p. 116)* Enables the collection and distribution of telemetry data for an RTI Connext application using RTI **Monitoring** (p. 1706) Library 2.0.

RTI **Monitoring** (p. 1706) Library 2.0 is **automatically enabled** under the following conditions:

- For RTI Connext applications that link with the RTI Connext shared libraries as long as the rtimonitoring2 shared library is in the library search path. There is no need to link with rtimonitoring2.
- For RTI Connext applications that link with the RTI Connext static libraries, including rtimonitoring2, as long as the application calls the API **RTI_Monitoring_initialize** (p. 599).

RTI **Monitoring** (p. 1706) Library 2.0 is **not automatically enabled** for RTI Connext RTI DDS Ping, RTI DDS Spy, RTI Monitor, RTI Admin Console nor RTI Observability Collector **Service** (p. 2350).

You can override the default behavior and control whether RTI **Monitoring** (p. 1706) Library 2.0 is enabled or disabled in two ways in order of precedence:

- **Environment variable:** Set `RTI_MONITORING2_ENABLE` to `true` or `false`. This is the simplest way to enable or disable the library without modifying your application or its configuration.
- **QoS policy:** Set this field to true or false programmatically or via XML.

The environment variable should not be changed at runtime. If you need to **enable** or **disable** RTI **Monitoring** (p. 1706) Library 2.0 at runtime, use this field instead.

Note: Enabling and disabling RTI **Monitoring** (p. 1706) Library 2.0 while DDS Entities are being created or deleted is not a safe operation. The entities created while RTI **Monitoring** (p. 1706) Library 2.0 is being enabled may not be monitored.

Categories

Mutable (p. 280), **Debuggability** (p. 172)

8.181.3.4 enable() [2/2]

```
bool rti::core::policy::Monitoring::enable ( ) const
```

Getter (see setter with the same name).

8.181.3.5 application_name() [1/3]

```
Monitoring & rti::core::policy::Monitoring::application_name (
            const  rti::core::optional_value< std::string > & the_application_name )
```

Sets the name of the resource that represents this RTI Connext application.

Parameters

the_application_name	An optional string. An unset value indicates that the application name will be automatically assigned. A set value will set an application name.

[default] Unset

When this member is set to a value, the resource identifier representing this application will be:

/applications/<application_name>

This is the resource identifier that will be used to send commands to this application from the RTI Observability Dashboards.

The application_name should be unique across the RTI Connext system; however, RTI **Monitoring** (p. 1706) Library 2.0 does not currently enforce uniqueness.

When this member is not set, RTI **Monitoring** (p. 1706) Library 2.0 will automatically assign a resource identifier with this format:

/applications/<host_name:process_id:uuid>

Categories

Mutable (p. 280), **Debuggability** (p. 172)

8.181.3.6 application_name() [2/3]

```
Monitoring & rti::core::policy::Monitoring::application_name (
            const char * the_application_name )
```

Sets the name of the resource that represents this RTI Connext application.

Parameters

the_application_name	The application name. Cannot be NULL.

When this member is set to a value, the resource identifier representing this application will be:

/applications/<application_name>

This is the resource identifier that will be used to send commands to this application from the RTI Observability Dashboards.

The application_name should be unique across the RTI Connext system; however, RTI **Monitoring** (p. 1706) Library 2.0 does not currently enforce uniqueness.

When this member is not set, RTI **Monitoring** (p. 1706) Library 2.0 will automatically assign a resource identifier with this format:

/applications/<host_name:process_id:uuid>

Categories

Mutable (p. 280), **Debuggability** (p. 172)

8.181.3.7 application_name() [3/3]

```
rti::core::optional_value< std::string > rti::core::policy::Monitoring::application_name ( )
const
```

Getter (see setter with the same name).

8.181.3.8 distribution_settings() [1/3]

```
Monitoring & rti::core::policy::Monitoring::distribution_settings (
            const  MonitoringDistributionSettings & the_distribution_settings )
```

<<***extension***>> *(p. 116)* Configures the distribution of telemetry data.

Categories

Mutable (p. 280), **Debuggability** (p. 172)

8.181.3.9 distribution_settings() [2/3]

```
const  MonitoringDistributionSettings & rti::core::policy::Monitoring::distribution_settings ( )
const
```

Gets the distribution settings by const-reference (see setter).

8.181.3.10 distribution_settings() [3/3]

```
MonitoringDistributionSettings & rti::core::policy::Monitoring::distribution_settings ( )
```

Gets the distribution settings by reference (see setter).

8.181.3.11 telemetry_data() [1/3]

```
Monitoring & rti::core::policy::Monitoring::telemetry_data (
            const  MonitoringTelemetryData & the_telemetry_data )
```

<<***extension***>> *(*p. *116)* Configures the telemetry data that will be distributed.

Categories

Mutable (p. 280), **Debuggability** (p. 172)

8.181.3.12 telemetry_data() [2/3]

```
const  MonitoringTelemetryData & rti::core::policy::Monitoring::telemetry_data ( ) const
```

Gets the telemetry data by const-reference (see setter).

8.181.3.13 telemetry_data() [3/3]

```
MonitoringTelemetryData & rti::core::policy::Monitoring::telemetry_data ( )
```

Gets the telemetry data by reference (see setter).

8.182 rti::core::MonitoringDedicatedParticipantSettings Class Reference

<<***extension***>> *(p. 354)* Configures the usage of a dedicated **dds::domain::DomainParticipant** (p. 1326) to distribute the RTI Connext application telemetry data.

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **MonitoringDedicatedParticipantSettings** ()

  Creates an instance with the default settings.
- **MonitoringDedicatedParticipantSettings** & **enable** (bool the_enable)

  <<***basic***>> *(p. 104)* <<***extension***>> *(p. 116) Enables the use of a dedicated* ***dds::domain::DomainParticipant*** *(p. 1326) to distribute the RTI Connext application telemetry data.*
- bool **enable** () const

  Getter (see setter with the same name).
- **MonitoringDedicatedParticipantSettings** & **domain_id** (int32_t the_domain_id)

  <<***basic***>> *(p. 104)* <<***extension***>> *(p. 116) The domain ID used in the creation of RTI Monitoring Library 2.0* ***dds::domain::DomainParticipant*** *(p. 1326).*
- int32_t **domain_id** () const

  Getter (see setter with the same name).
- **MonitoringDedicatedParticipantSettings** & **participant_qos_profile_name** (const **rti::core::optional_value**< std::string > &the_participant_qos_profile_name)

  Sets the fully qualified name of the profile used to configure the ***dds::domain::DomainParticipant*** *(p. 1326) that will be used to distribute telemetry data.*
- **MonitoringDedicatedParticipantSettings** & **participant_qos_profile_name** (const char *the_participant_qos_profile_name)

  Sets the fully qualified name of the profile used to configure the ***dds::domain::DomainParticipant*** *(p. 1326) that will be used to distribute telemetry data.*
- **rti::core::optional_value**< std::string > **participant_qos_profile_name** () const

  Getter (see setter with the same name).
- **MonitoringDedicatedParticipantSettings** & **collector_initial_peers** (const **dds::core::StringSeq** &the_collector_initial_peers)

  Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the ***rti::core::MonitoringDistributionSettings::dedicated_participant*** *(p. 1719).*
- **dds::core::StringSeq collector_initial_peers** () const

  Getter (see setter with the same name)

8.182.1 Detailed Description

<<***extension***>> *(p. 354)* Configures the usage of a dedicated **dds::domain::DomainParticipant** (p. 1326) to distribute the RTI Connext application telemetry data.

8.182.2 Constructor & Destructor Documentation

8.182.2.1 MonitoringDedicatedParticipantSettings()

```
rti::core::MonitoringDedicatedParticipantSettings::MonitoringDedicatedParticipantSettings ( )
[inline]
```

Creates an instance with the default settings.

8.182.3 Member Function Documentation

8.182.3.1 enable() [1/2]

```
MonitoringDedicatedParticipantSettings & rti::core::MonitoringDedicatedParticipantSettings↩
::enable (
            bool the_enable )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Enables the use of a dedicated **dds::domain::DomainParticipant** (p. 1326) to distribute the RTI Connext application telemetry data.

Setting this value to false is not currently supported.

[default] true

Categories

Mutable (p. 280), **Debuggability** (p. 172), **QoS Profile Management** (p. 220)

8.182.3.2 enable() [2/2]

```
bool rti::core::MonitoringDedicatedParticipantSettings::enable ( ) const
```

Getter (see setter with the same name).

8.182.3.3 domain_id() [1/2]

```
MonitoringDedicatedParticipantSettings & rti::core::MonitoringDedicatedParticipantSettings↩
::domain_id (
            int32_t the_domain_id )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* The domain ID used in the creation of RTI Monitoring Library 2.0 **dds::domain::DomainParticipant** (p. 1326).

By default, the RTI Monitoring Library 2.0 DomainParticipant is created in domain 101 with the domain tag 'RTI_o11y'. The domain tag provides further isolation in case the application also uses domain 101.

The domain ID can be changed using this field. Changing the domain tag is not recommended. However, if you need to change it, you must do so by configuring the domain tag in the **dds::domain::qos::DomainParticipantQos** (p. 1384) of the profile set in **rti::core::MonitoringDedicatedParticipantSettings::participant_qos_profile_name** (p. 1714).

[default] 101

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Discovery** (p. 177), **Discovery Isolation** (p. 183)

8.182.3.4 domain_id() [2/2]

```
int32_t rti::core::MonitoringDedicatedParticipantSettings::domain_id ( ) const
```

Getter (see setter with the same name).

8.182.3.5 participant_qos_profile_name() [1/3]

```
MonitoringDedicatedParticipantSettings & rti::core::MonitoringDedicatedParticipantSettings↩
::participant_qos_profile_name (
            const  rti::core::optional_value< std::string > & the_participant_qos_profile_name )
```

Sets the fully qualified name of the profile used to configure the **dds::domain::DomainParticipant** (p. 1326) that will be used to distribute telemetry data.

Parameters

the_participant_qos_profile_name	An optional string. An unset value indicates that the RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_monitoring2()** (p. 479).

[default] Unset

If not set (the default value) then RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_↩ monitoring2()** (p. 479).

Categories

Mutable (p. 280), **Debuggability** (p. 172), **QoS Profile Management** (p. 220)

8.182.3.6 participant_qos_profile_name() [2/3]

```
MonitoringDedicatedParticipantSettings & rti::core::MonitoringDedicatedParticipantSettings↩
::participant_qos_profile_name (
            const char * the_participant_qos_profile_name )
```

Sets the fully qualified name of the profile used to configure the **dds::domain::DomainParticipant** (p. 1326) that will be used to distribute telemetry data.

Parameters

the_participant_qos_profile_name	The profile name. Cannot be NULL.

If not set (the default value) then RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_↩ monitoring2()** (p. 479).

Categories

Mutable (p. 280), **Debuggability** (p. 172), **QoS Profile Management** (p. 220)

8.182.3.7 participant_qos_profile_name() [3/3]

```
rti::core::optional_value< std::string > rti::core::MonitoringDedicatedParticipantSettings↩
::participant_qos_profile_name ( ) const
```

Getter (see setter with the same name).

8.182.3.8 collector_initial_peers() [1/2]

```
MonitoringDedicatedParticipantSettings & rti::core::MonitoringDedicatedParticipantSettings↩
::collector_initial_peers (
            const  dds::core::StringSeq & the_collector_initial_peers )
```

Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **rti::core::MonitoringDistributionSettings::dedicated_participant** (p. 1719).

Parameters

the_collector_initial_peers	The initial peers.

These initial peers should correspond with the RTI Observability Collector Service with which RTI Monitoring Library 2.0 has to communicate. The collector_initial_peers works the same as initial_peers for other DomainParticipants, except that it allows you to easily specify the initial peer(s) for the RTI Monitoring Library 2.0 **dds::domain::DomainParticipant** (p. 1326), which may have different initial peer(s) than those used by your application.

If no collector_initial_peers are specified, or if it is explicitly set to an empty list, the **rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers)** (p. 1268) list of **rti::core::MonitoringDedicatedParticipantSettings::participant_qos_profile_name** (p. 1714) will be used as the initial peers of **rti::core::MonitoringDistributionSettings::dedicated_participant** (p. 1719).

[default] An empty sequence.

See also

rti::core::policy::Discovery::initial_peers(const dds::core::StringSeq & the_initial_peers) (p. 1268) for further information about initial peers.

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Discovery** (p. 177), **Discovery Isolation** (p. 183)

8.182.3.9 collector_initial_peers() [2/2]

```
dds::core::StringSeq rti::core::MonitoringDedicatedParticipantSettings::collector_initial_peers (
) const
```

Getter (see setter with the same name)

8.183 rti::core::MonitoringDistributionSettings Class Reference

<<***extension***>> *(p. 354)* Configures the distribution of telemetry data.

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **MonitoringDistributionSettings** ()

  Creates an instance with the default settings.
- **MonitoringDistributionSettings** & **publisher_qos_profile_name** (const **rti::core::optional_value**< std::string > &the_publisher_qos_profile_name)

  Sets the fully qualified name of the profile used to configure the Publishers that distribute telemetry data.
- **MonitoringDistributionSettings** & **publisher_qos_profile_name** (const char *the_publisher_qos_profile_name)

  Sets the fully qualified name of the profile used to configure the Publishers that distribute telemetry data.
- **rti::core::optional_value**< std::string > **publisher_qos_profile_name** () const

  Getter (see setter with the same name).
- **MonitoringDistributionSettings** & **dedicated_participant** (const **MonitoringDedicatedParticipantSettings** &the_dedicated_participant)

  <<***extension***>> *(*p. *116) Configures the use of a dedicated* ***dds::domain::DomainParticipant*** *(*p. *1326) to distribute the RTI Connext application telemetry data.*
- const **MonitoringDedicatedParticipantSettings** & **dedicated_participant** () const

  Gets the distribution settings by const-reference (see setter).
- **MonitoringDedicatedParticipantSettings** & **dedicated_participant** ()

  Gets the distribution settings by reference (see setter).
- **MonitoringDistributionSettings** & **event_settings** (const **MonitoringEventDistributionSettings** &the_event_settings)

  <<***extension***>> *(*p. *116) Configures the distribution of event metrics.*
- const **MonitoringEventDistributionSettings** & **event_settings** () const

  Gets the distribution settings by const-reference (see setter).
- **MonitoringEventDistributionSettings** & **event_settings** ()

  Gets the distribution settings by reference (see setter).
- **MonitoringDistributionSettings** & **periodic_settings** (const **MonitoringPeriodicDistributionSettings** &the_periodic_settings)

  <<***extension***>> *(*p. *116) Configures the distribution of periodic metrics.*
- const **MonitoringPeriodicDistributionSettings** & **periodic_settings** () const

  Gets the distribution settings by const-reference (see setter).
- **MonitoringPeriodicDistributionSettings** & **periodic_settings** ()

  Gets the distribution settings by reference (see setter).
- **MonitoringDistributionSettings** & **logging_settings** (const **MonitoringLoggingDistributionSettings** &the_logging_settings)

  <<***extension***>> *(*p. *116) Configures the distribution of logging messages.*
- const **MonitoringLoggingDistributionSettings** & **logging_settings** () const

  Gets the distribution settings by const-reference (see setter).
- **MonitoringLoggingDistributionSettings** & **logging_settings** ()

  Gets the distribution settings by reference (see setter).

8.183.1 Detailed Description

<<***extension***>> *(*p. *354)* Configures the distribution of telemetry data.

8.183.2 Constructor & Destructor Documentation

8.183.2.1 MonitoringDistributionSettings()

```
rti::core::MonitoringDistributionSettings::MonitoringDistributionSettings ( )  [inline]
```

Creates an instance with the default settings.

8.183.3 Member Function Documentation

8.183.3.1 publisher_qos_profile_name() [1/3]

```
MonitoringDistributionSettings & rti::core::MonitoringDistributionSettings::publisher_qos_↩
profile_name (
            const  rti::core::optional_value< std::string > & the_publisher_qos_profile_name )
```

Sets the fully qualified name of the profile used to configure the Publishers that distribute telemetry data.

Parameters

the_publisher_qos_profile_name	An optional string. An unset value indicates that the RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_monitoring2()** (p. 479).

[default] Unset

There is one Publisher for each telemetry data **dds::topic::Topic** (p. 2474): MONITORING_PERIODIC_TOPIC_NAME, MONITORING_EVENT_TOPIC_NAME, and MONITORING_LOGGING_TOPIC_NAME.

If not set (the default value) then RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_↩ monitoring2()** (p. 479).

Categories

Mutable (p. 280), **Debuggability** (p. 172), **QoS Profile Management** (p. 220)

8.183.3.2 publisher_qos_profile_name() [2/3]

```
MonitoringDistributionSettings & rti::core::MonitoringDistributionSettings::publisher_qos_↩
profile_name (
            const char * the_publisher_qos_profile_name )
```

Sets the fully qualified name of the profile used to configure the Publishers that distribute telemetry data.

Parameters

the_publisher_qos_profile_name	The profile name. Cannot be NULL.

There is one Publisher for each telemetry data **dds::topic::Topic** (p. 2474): MONITORING_PERIODIC_TOPIC_NAME, MONITORING_EVENT_TOPIC_NAME, and MONITORING_LOGGING_TOPIC_NAME.

If not set (the default value) then RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_↩ monitoring2()** (p. 479).

Categories

Mutable (p. 280), **Debuggability** (p. 172), **QoS Profile Management** (p. 220)

8.183.3.3 publisher_qos_profile_name() [3/3]

```
rti::core::optional_value< std::string > rti::core::MonitoringDistributionSettings::publisher_↩
qos_profile_name ( ) const
```

Getter (see setter with the same name).

8.183.3.4 dedicated_participant() [1/3]

```
MonitoringDistributionSettings & rti::core::MonitoringDistributionSettings::dedicated_participant
(
            const  MonitoringDedicatedParticipantSettings & the_dedicated_participant )
```

<<***extension***>> *(p. 116)* Configures the use of a dedicated **dds::domain::DomainParticipant** (p. 1326) to distribute the RTI Connext application telemetry data.

Categories

Mutable (p. 280), **Debuggability** (p. 172), **QoS Profile Management** (p. 220)

8.183.3.5 dedicated_participant() [2/3]

```
const  MonitoringDedicatedParticipantSettings & rti::core::MonitoringDistributionSettings::dedicated↩
_participant ( ) const
```

Gets the distribution settings by const-reference (see setter).

8.183.3.6 dedicated_participant() [3/3]

```
MonitoringDedicatedParticipantSettings & rti::core::MonitoringDistributionSettings::dedicated_↩
participant ( )
```

Gets the distribution settings by reference (see setter).

8.183.3.7 event_settings() [1/3]

```
MonitoringDistributionSettings & rti::core::MonitoringDistributionSettings::event_settings (
            const  MonitoringEventDistributionSettings & the_event_settings )
```

<<***extension***>> *(p. 116)* Configures the distribution of event metrics.

Categories

Mutable (p. 280), **Debuggability** (p. 172)

8.183.3.8 event_settings() [2/3]

```
const  MonitoringEventDistributionSettings & rti::core::MonitoringDistributionSettings::event_↩
settings ( ) const
```

Gets the distribution settings by const-reference (see setter).

8.183.3.9 event_settings() [3/3]

```
MonitoringEventDistributionSettings & rti::core::MonitoringDistributionSettings::event_settings (
)
```

Gets the distribution settings by reference (see setter).

8.183.3.10 periodic_settings() [1/3]

```
MonitoringDistributionSettings & rti::core::MonitoringDistributionSettings::periodic_settings (
            const  MonitoringPeriodicDistributionSettings & the_periodic_settings )
```

<<***extension***>> *(p. 116)* Configures the distribution of periodic metrics.

Categories

Mutable (p. 280), **Debuggability** (p. 172)

8.183.3.11 periodic_settings() [2/3]

```
const  MonitoringPeriodicDistributionSettings & rti::core::MonitoringDistributionSettings::periodic↩
_settings ( ) const
```

Gets the distribution settings by const-reference (see setter).

8.183.3.12 periodic_settings() [3/3]

```
MonitoringPeriodicDistributionSettings & rti::core::MonitoringDistributionSettings::periodic_↩
settings ( )
```

Gets the distribution settings by reference (see setter).

8.183.3.13 logging_settings() [1/3]

```
MonitoringDistributionSettings & rti::core::MonitoringDistributionSettings::logging_settings (
            const  MonitoringLoggingDistributionSettings & the_logging_settings )
```

<<***extension***>> *(p. 116)* Configures the distribution of logging messages.

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Logging** (p. 201)

8.183.3.14 logging_settings() [2/3]

```
const MonitoringLoggingDistributionSettings & rti::core::MonitoringDistributionSettings::logging↩
_settings ( ) const
```

Gets the distribution settings by const-reference (see setter).

8.183.3.15 logging_settings() [3/3]

```
MonitoringLoggingDistributionSettings & rti::core::MonitoringDistributionSettings::logging_↩
settings ( )
```

Gets the distribution settings by reference (see setter).

8.184 rti::core::MonitoringEventDistributionSettings Class Reference

<<***extension***>> *(*p. *354)* Configures the distribution of event metrics.

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **MonitoringEventDistributionSettings** ()

  Creates an instance with the default settings.
- **MonitoringEventDistributionSettings** & **concurrency_level** (uint32_t the_concurrency_level)

  <<***extension***>> *(*p. *116) Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.*
- uint32_t **concurrency_level** () const

  Getter (see setter with the same name).
- **MonitoringEventDistributionSettings** & **datawriter_qos_profile_name** (const **rti::core::optional_value**< std::string > &the_datawriter_qos_profile_name)

  Sets The fully qualified name of the profile used to configure the ***dds::pub::DataWriter*** *(*p. *1138) that distributes event metrics.*
- **MonitoringEventDistributionSettings** & **datawriter_qos_profile_name** (const char *the_datawriter_qos_↩ profile_name)

  Sets The fully qualified name of the profile used to configure the ***dds::pub::DataWriter*** *(*p. *1138) that distributes event metrics.*
- **rti::core::optional_value**< std::string > **datawriter_qos_profile_name** () const

  Getter (see setter with the same name).
- **MonitoringEventDistributionSettings** & **thread** (const **rti::core::ThreadSettings** &the_thread)

  <<***extension***>> *(*p. *116) The settings of the event metric thread.*
- const **rti::core::ThreadSettings** & **thread** () const

  Getter (see setter with the same name).
- **rti::core::ThreadSettings** & **thread** ()

  Getter (see setter with the same name).
- **MonitoringEventDistributionSettings** & **publication_period** (const **dds::core::Duration** &the_publication↩ _period)

  <<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116) Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.*
- **dds::core::Duration publication_period** () const

  Getter (see setter with the same name)

8.184.1 Detailed Description

<<***extension***>> *(p. 354)* Configures the distribution of event metrics.

Event metrics are provided to RTI Monitoring Library 2.0 when they change.

For example, if the liveliness of a **dds::pub::DataWriter** (p. 1138) is lost, a matching **dds::sub::DataReader** (p. 976) will push the new value of **dds::core::status::LivelinessChangedStatus** (p. 1652) to RTI Monitoring Library 2.0 so that the liveliness status change can be distributed.

There are three kinds of event metrics:

- **Configuration metrics:** Provided to RTI Monitoring Library 2.0 by pushing changes to QoS policies.
- **Status metrics:** Provided to RTI Monitoring Library 2.0 by pushing changes to the event statuses such as **dds::core::status::LivelinessChangedStatus** (p. 1652).
- **Resource metrics:** Provided to RTI Monitoring Library 2.0 when a resource (such as **dds::pub::DataWriter** (p. 1138)) is created or deleted.

The event metrics that will be distributed for an observable resource can be configured with **rti::core::MonitoringTelemetryData::metrics** (p. 1745).

8.184.2 Constructor & Destructor Documentation

8.184.2.1 MonitoringEventDistributionSettings()

```
rti::core::MonitoringEventDistributionSettings::MonitoringEventDistributionSettings ( )  [inline]
```

Creates an instance with the default settings.

8.184.3 Member Function Documentation

8.184.3.1 concurrency_level() [1/2]

```
MonitoringEventDistributionSettings & rti::core::MonitoringEventDistributionSettings::concurrency_level (
            uint32_t the_concurrency_level )
```

<<***extension***>> *(p. 116)* Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.

With a concurrency_level of one, all the event metrics pushed to RTI Monitoring Library 2.0 will be stored in a single queue protected by a single mutex.

With a concurrency_level of 'n', RTI Monitoring Library 2.0 will create 'n' queues for event metrics, each queue protected by its own mutex. Each resource (e.g, a **dds::sub::DataReader** (p. 976)) will be associated with one of the queues when the resource is registered with RTI Monitoring Library 2.0. Therefore, all the event metrics for a single resource always go to the same queue.

The event metrics for two resources associated with different event queues can be pushed in parallel. This is why a higher concurrency_level provides more concurrency.

The event metrics added to the event queues are processed by a single thread configured using **rti::core::MonitoringEventDistributionSettings::thread** (p. 1726).

[default] 5

[range] [1, 100]

Categories

Mutable (p. 280), **Concurrency** (p. 164), **Debuggability** (p. 172)

8.184.3.2 concurrency_level() [2/2]

```
uint32_t rti::core::MonitoringEventDistributionSettings::concurrency_level ( ) const
```

Getter (see setter with the same name).

8.184.3.3 datawriter_qos_profile_name() [1/3]

```
MonitoringEventDistributionSettings & rti::core::MonitoringEventDistributionSettings::datawriter_qos_profile_name (
            const  rti::core::optional_value< std::string > & the_datawriter_qos_profile_name )
```

Sets The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes event metrics.

Parameters

the_datawriter_qos_profile_name	An optional string. An unset value indicates that the RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_monitoring2()** (p. 479).

[default] Unset

The **dds::pub::DataWriter** (p. 1138) Topic is MONITORING_EVENT_TOPIC_NAME.

If null (the default value), then RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_↩ monitoring2()** (p. 479).

[default] null

Categories

Mutable (p. 280), **Debuggability** (p. 172), **QoS Profile Management** (p. 220)

8.184.3.4 datawriter_qos_profile_name() [2/3]

```
MonitoringEventDistributionSettings & rti::core::MonitoringEventDistributionSettings::datawriter↩
_qos_profile_name (
            const char * the_datawriter_qos_profile_name )
```

Sets The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes event metrics.

Parameters

the_datawriter_qos_profile_name	The profile name. Cannot be NULL.

The **dds::pub::DataWriter** (p. 1138) Topic is MONITORING_EVENT_TOPIC_NAME.

If null (the default value), then RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_↩ monitoring2()** (p. 479).

[default] null

Categories

Mutable (p. 280), **Debuggability** (p. 172), **QoS Profile Management** (p. 220)

8.184.3.5 datawriter_qos_profile_name() [3/3]

```
rti::core::optional_value< std::string > rti::core::MonitoringEventDistributionSettings::datawriter_qos_profile_name ( ) const
```

Getter (see setter with the same name).

8.184.3.6 thread() [1/3]

```
MonitoringEventDistributionSettings & rti::core::MonitoringEventDistributionSettings::thread (
            const  rti::core::ThreadSettings & the_thread )
```

<<***extension***>> *(*p. *116)* The settings of the event metric thread.

The event metric thread periodically publishes the event metrics pushed into RTI Monitoring Library 2.0 event metric queues after they change their values.

The thread runs at the period configured using **rti::core::MonitoringEventDistributionSettings::publication_period** (p. 1726).

[default] DDS_THREAD_SETTINGS_DEFAULT

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Concurrency** (p. 164)

8.184.3.7 thread() [2/3]

```
const  rti::core::ThreadSettings & rti::core::MonitoringEventDistributionSettings::thread ( )
const
```

Getter (see setter with the same name).

8.184.3.8 thread() [3/3]

```
rti::core::ThreadSettings & rti::core::MonitoringEventDistributionSettings::thread ( )
```

Getter (see setter with the same name).

8.184.3.9 publication_period() [1/2]

```
MonitoringEventDistributionSettings & rti::core::MonitoringEventDistributionSettings::publication_period (
            const  dds::core::Duration & the_publication_period )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.

With a period of 0 seconds, changes to event metrics will be published immediately after they are pushed into RTI Monitoring Library 2.0.

[default] 1 second

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Data Availability** (p. 167)

8.184.3.10 publication_period() [2/2]

```
dds::core::Duration rti::core::MonitoringEventDistributionSettings::publication_period ( ) const
```

Getter (see setter with the same name)

8.185 rti::core::MonitoringLoggingDistributionSettings Class Reference

<<***extension***>> *(*p. *354)* Configures the distribution of log messages.

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **MonitoringLoggingDistributionSettings** ()
  Creates an instance with the default settings.
- **MonitoringLoggingDistributionSettings** & **concurrency_level** (uint32_t the_concurrency_level)
  <<***extension***>> *(*p. *116) Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.*
- uint32_t **concurrency_level** () const
  Getter (see setter with the same name).
- **MonitoringLoggingDistributionSettings** & **max_historical_logs** (uint32_t the_max_historical_logs)
  <<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116) The number of log messages that RTI Monitoring Library 2.0 will keep as history.*
- uint32_t **max_historical_logs** () const
  Getter (see setter with the same name).

- **MonitoringLoggingDistributionSettings** & **datawriter_qos_profile_name** (const **rti::core::optional_value**< std::string > &the_datawriter_qos_profile_name)

  *Sets The fully qualified name of the profile used to configure the **dds::pub::DataWriter*** (p. *1138*) *that distributes log messages.*
- **MonitoringLoggingDistributionSettings** & **datawriter_qos_profile_name** (const char *the_datawriter_qos_profile_name)

  *Sets The fully qualified name of the profile used to configure the **dds::pub::DataWriter*** (p. *1138*) *that distributes log messages.*
- **rti::core::optional_value**< std::string > **datawriter_qos_profile_name** () const

  Getter (see setter with the same name).
- **MonitoringLoggingDistributionSettings** & **thread** (const **rti::core::ThreadSettings** &the_thread)

  <<***extension***>> (p. *116*) *The settings of the logging thread.*
- const **rti::core::ThreadSettings** & **thread** () const

  Getter (see setter with the same name).
- **rti::core::ThreadSettings** & **thread** ()

  Getter (see setter with the same name).
- **MonitoringLoggingDistributionSettings** & **publication_period** (const **dds::core::Duration** &the_publication_period)

  <<***basic***>> (p. *104*) <<***extension***>> (p. *116*) *Period at which the logging thread publishes log messages.*
- **dds::core::Duration publication_period** () const

  Getter (see setter with the same name)

8.185.1 Detailed Description

<<***extension***>> (p. *354*) Configures the distribution of log messages.

Log messages are pushed into RTI Monitoring Library 2.0 and published by the logging thread.

The logging thread only publishes a log message with a Syslog level smaller than or equal to the forwarding level of the **rti::config::LogFacility** (p. 459) associated with the log message.

The default value of the forwarding level for all facilities is **rti::config::SyslogVerbosity::warning** (p. 458). This value can be changed with the **rti::core::MonitoringTelemetryData::logs** (p. 1746) QoS Policy or by sending a command to RTI Monitoring Library 2.0.

In this release, commands can only be sent from the RTI Observability Dashboards.

RTI Monitoring Library 2.0 can be configured to keep a history of log messages for later distribution when a log snapshot is requested by a RTI Observability Collector Service.

The log messages maintained in the history are the last 'n' messages published by the logging thread (where 'n' is the value of **rti::core::MonitoringLoggingDistributionSettings::max_historical_logs** (p. 1729)).

8.185.2 Constructor & Destructor Documentation

8.185.2.1 MonitoringLoggingDistributionSettings()

```
rti::core::MonitoringLoggingDistributionSettings::MonitoringLoggingDistributionSettings ( )  [inline]
```

Creates an instance with the default settings.

8.185.3 Member Function Documentation

8.185.3.1 concurrency_level() [1/2]

```
MonitoringLoggingDistributionSettings & rti::core::MonitoringLoggingDistributionSettings::concurrency_level (
            uint32_t the_concurrency_level )
```

<<***extension***>> *(*p. *116)* Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.

With a concurrency_level of one, all the log messages pushed to RTI Monitoring Library 2.0 will be stored into a single queue protected by a single mutex.

With a concurrency_level of 'n', RTI Monitoring Library 2.0 will create 'n' queues for log messages, each queue protected by its own mutex.

The log messages generated by a single thread will always be pushed to the same queue. The log messages for two threads associated with different log queues can be pushed in parallel. This is why a higher concurrency_level provides more concurrency.

The log messages added to the log queues are processed and published by a single thread configured using **rti::core::MonitoringLoggingDistributionSettings::thread** (p. 1731).

[default] 5

[range] [1, 100]

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Logging** (p. 201), **Concurrency** (p. 164)

8.185.3.2 concurrency_level() [2/2]

```
uint32_t rti::core::MonitoringLoggingDistributionSettings::concurrency_level ( ) const
```

Getter (see setter with the same name).

8.185.3.3 max_historical_logs() [1/2]

```
MonitoringLoggingDistributionSettings & rti::core::MonitoringLoggingDistributionSettings::max_↩
historical_logs (
            uint32_t the_max_historical_logs )
```

<<***basic***>> *(p. 104)* <<***extension***>> *(p. 116)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.

RTI Monitoring Library 2.0 will keep as history the last `max_historical_logs` published messages.

A value of 0 means that RTI Monitoring Library 2.0 should not keep any history.

[default] 128

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Logging** (p. 201), **Middleware Memory Management** (p. 204)

8.185.3.4 max_historical_logs() [2/2]

```
uint32_t rti::core::MonitoringLoggingDistributionSettings::max_historical_logs ( ) const
```

Getter (see setter with the same name).

8.185.3.5 datawriter_qos_profile_name() [1/3]

```
MonitoringLoggingDistributionSettings & rti::core::MonitoringLoggingDistributionSettings::datawriter↩
_qos_profile_name (
            const  rti::core::optional_value< std::string > & the_datawriter_qos_profile_name )
```

Sets The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes log messages.

Parameters

the_datawriter_qos_profile_name	An optional string. An unset value indicates that the RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_monitoring2()** (p. 479).

[default] Unset

The **dds::pub::DataWriter** (p. 1138) Topic is MONITORING_LOGGING_TOPIC_NAME.

If null (the default value), then RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_↩ monitoring2()** (p. 479).

[default] null

Categories

Mutable (p. 280), **Debuggability** (p. 172), **QoS Profile Management** (p. 220), **Logging** (p. 201)

8.185.3.6 datawriter_qos_profile_name() [2/3]

```
MonitoringLoggingDistributionSettings & rti::core::MonitoringLoggingDistributionSettings::datawriter↩
_qos_profile_name (
            const char * the_datawriter_qos_profile_name )
```

Sets The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes log messages.

Parameters

the_datawriter_qos_profile_name	The profile name. Cannot be NULL.

The **dds::pub::DataWriter** (p. 1138) Topic is MONITORING_LOGGING_TOPIC_NAME.

If null (the default value), then RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_↩ monitoring2()** (p. 479).

[default] null

Categories

Mutable (p. 280), **Debuggability** (p. 172), **QoS Profile Management** (p. 220), **Logging** (p. 201)

8.185.3.7 datawriter_qos_profile_name() [3/3]

```
rti::core::optional_value< std::string > rti::core::MonitoringLoggingDistributionSettings↩
::datawriter_qos_profile_name ( ) const
```

Getter (see setter with the same name).

8.185.3.8 thread() [1/3]

```
MonitoringLoggingDistributionSettings & rti::core::MonitoringLoggingDistributionSettings::thread
(
            const  rti::core::ThreadSettings & the_thread )
```

<<***extension***>> *(*p. *116)* The settings of the logging thread.

The logging thread periodically publishes the log messages pushed into RTI Monitoring Library 2.0 log message queues after they are generated.

The thread runs at the period configured using **rti::core::MonitoringLoggingDistributionSettings::publication_period** (p. 1732).

[default] DDS_THREAD_SETTINGS_DEFAULT

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Concurrency** (p. 164), **Logging** (p. 201)

8.185.3.9 thread() [2/3]

```
const  rti::core::ThreadSettings & rti::core::MonitoringLoggingDistributionSettings::thread ( )
const
```

Getter (see setter with the same name).

8.185.3.10 thread() [3/3]

```
rti::core::ThreadSettings & rti::core::MonitoringLoggingDistributionSettings::thread ( )
```

Getter (see setter with the same name).

8.185.3.11 publication_period() [1/2]

```
MonitoringLoggingDistributionSettings & rti::core::MonitoringLoggingDistributionSettings::publication
_period (
            const  dds::core::Duration & the_publication_period )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Period at which the logging thread publishes log messages.

With a period of 0 seconds, log messages will be published immediately after they are pushed into RTI Monitoring Library 2.0.

[default] 1 second

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Data Availability** (p. 167), **Logging** (p. 201)

8.185.3.12 publication_period() [2/2]

```
dds::core::Duration rti::core::MonitoringLoggingDistributionSettings::publication_period ( )
const
```

Getter (see setter with the same name)

8.186 rti::core::MonitoringLoggingForwardingSettings Class Reference

<<***extension***>> *(*p. *354)* Configures the forwarding levels of log messages for the different **rti::config::LogFacility** (p. 459).

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **MonitoringLoggingForwardingSettings** & **middleware_forwarding_level** (const **rti::config::Syslog↩ Verbosity** &middleware_forwarding_level)

  *Sets the forwarding log level for **rti::config::LogFacility::middleware** (*p. *459). Log messages with **rti::config::Log↩ Facility::middleware** (*p. *459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.*
- **rti::config::SyslogVerbosity middleware_forwarding_level** () const

  Gets the middleware forwarding level by const-reference (see setter).
- **MonitoringLoggingForwardingSettings** & **security_event_forwarding_level** (const **rti::config::Syslog↩ Verbosity** &security_event_forwarding_level)

  *Sets the forwarding log level for **rti::config::LogFacility::security_event** (*p. *459). Log messages with **rti::config::Log↩ Facility::security_event** (*p. *459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.*
- **rti::config::SyslogVerbosity security_event_forwarding_level** () const

  Gets the security event forwarding level by const-reference (see setter).
- **MonitoringLoggingForwardingSettings** & **service_forwarding_level** (const **rti::config::SyslogVerbosity** &service_forwarding_level)

  *Sets the forwarding log level for **rti::config::LogFacility::service** (*p. *459). Log messages with **rti::config::LogFacility↩ ::service** (*p. *459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.*
- **rti::config::SyslogVerbosity service_forwarding_level** () const

  Gets the service forwarding level by const-reference (see setter).
- **MonitoringLoggingForwardingSettings** & **user_forwarding_level** (const **rti::config::SyslogVerbosity** &user_forwarding_level)

  *Sets the forwarding log level for **rti::config::LogFacility::user** (*p. *459). Log messages with **rti::config::LogFacility↩ ::user** (*p. *459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.*
- **rti::config::SyslogVerbosity user_forwarding_level** () const

  Gets the user forwarding level by const-reference (see setter).

8.186.1 Detailed Description

<<***extension***>> *(*p. *354)* Configures the forwarding levels of log messages for the different **rti::config::LogFacility** (p. 459).

8.186.2 Member Function Documentation

8.186.2.1 middleware_forwarding_level() [1/2]

```
MonitoringLoggingForwardingSettings & rti::core::MonitoringLoggingForwardingSettings::middleware_forwarding_level (
            const  rti::config::SyslogVerbosity & middleware_forwarding_level )
```

Sets the forwarding log level for **rti::config::LogFacility::middleware** (p. 459). Log messages with **rti::config::LogFacility::middleware** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

Parameters

middleware_forwarding_level	**rti::config::SyslogVerbosity** (p. 458) level.

[default] rti::config::SyslogVerbosity::warning (p. 458)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **rti::config::Verbosity_def::warning** (p. 2614), may affect performance due to the large amount of messages produced.

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Logging** (p. 201)

8.186.2.2 middleware_forwarding_level() [2/2]

```
rti::config::SyslogVerbosity rti::core::MonitoringLoggingForwardingSettings::middleware_forwarding_level ( ) const
```

Gets the middleware forwarding level by const-reference (see setter).

8.186.2.3 security_event_forwarding_level() [1/2]

```
MonitoringLoggingForwardingSettings & rti::core::MonitoringLoggingForwardingSettings::security_event_forwarding_level (
            const  rti::config::SyslogVerbosity & security_event_forwarding_level )
```

Sets the forwarding log level for **rti::config::LogFacility::security_event** (p. 459). Log messages with **rti::config::LogFacility::security_event** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

Parameters

security_event_forwarding_level	**rti::config::SyslogVerbosity** (p. 458) level.

[default] rti::config::SyslogVerbosity::warning (p. 458)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **rti::config::Verbosity_def::warning** (p. 2614), may affect performance due to the large amount of messages produced.

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Logging** (p. 201), **Security** (p. 235)

8.186.2.4 security_event_forwarding_level() [2/2]

```
rti::config::SyslogVerbosity rti::core::MonitoringLoggingForwardingSettings::security_event_forwarding_level ( ) const
```

Gets the security event forwarding level by const-reference (see setter).

8.186.2.5 service_forwarding_level() [1/2]

```
MonitoringLoggingForwardingSettings & rti::core::MonitoringLoggingForwardingSettings::service_forwarding_level (
            const  rti::config::SyslogVerbosity & service_forwarding_level )
```

Sets the forwarding log level for **rti::config::LogFacility::service** (p. 459). Log messages with **rti::config::LogFacility::service** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

Parameters

service_forwarding_level	**rti::config::SyslogVerbosity** (p. 458) level.

[default] rti::config::SyslogVerbosity::warning (p. 458)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **rti::config::Verbosity_def::warning** (p. 2614), may affect performance due to the large amount of messages produced.

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Logging** (p. 201)

8.186.2.6 service_forwarding_level() [2/2]

```
rti::config::SyslogVerbosity rti::core::MonitoringLoggingForwardingSettings::service_forwarding_
level ( ) const
```

Gets the service forwarding level by const-reference (see setter).

8.186.2.7 user_forwarding_level() [1/2]

```
MonitoringLoggingForwardingSettings & rti::core::MonitoringLoggingForwardingSettings::user_
forwarding_level (
            const  rti::config::SyslogVerbosity & user_forwarding_level )
```

Sets the forwarding log level for **rti::config::LogFacility::user** (p. 459). Log messages with **rti::config::LogFacility::user** (p. 459) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

Parameters

user_forwarding_level	**rti::config::SyslogVerbosity** (p. 458) level.

[default] rti::config::SyslogVerbosity::warning (p. 458)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **rti::config::Verbosity_def::warning** (p. 2614), may affect performance due to the large amount of messages produced.

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Logging** (p. 201)

8.186.2.8 user_forwarding_level() [2/2]

```
rti::config::SyslogVerbosity rti::core::MonitoringLoggingForwardingSettings::user_forwarding_
level ( ) const
```

Gets the user forwarding level by const-reference (see setter).

8.187 rti::core::MonitoringMetricSelection Class Reference

<<***extension***>> *(p. 354)* Configures event and periodic metrics collection and distribution for a specific set of observable resources.

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **MonitoringMetricSelection** ()=default

  Creates an instance with the default settings.
- **MonitoringMetricSelection** & **resource_selection** (const std::string &the_resource_selection)

  An expression pattern that selects a subset of resources by matching the pattern to the resource names. ***rti::core::MonitoringMetricSelection::enabled_metrics_selection*** *(p. 1738) and* ***rti::core::MonitoringMetricSelection::disabled_metrics_selection*** *(p. 1739) are applied to this subset of resources.*
- std::string **resource_selection** () const

  Getter (see setter with the same name).
- **MonitoringMetricSelection** & **enabled_metrics_selection** (const **dds::core::StringSeq** &the_enabled_metrics_selection)

  Sets A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by ***rti::core::MonitoringMetricSelection::resource_selection*** *(p. 1737).*
- **dds::core::StringSeq enabled_metrics_selection** () const

  Getter (see setter with the same name)
- **MonitoringMetricSelection** & **disabled_metrics_selection** (const **dds::core::StringSeq** &the_disabled_metrics_selection)

  Sets A sequence of POSIX fnmatch patterns that match the names of the metrics that should not be collected and distributed for the observable resources selected by ***rti::core::MonitoringMetricSelection::resource_selection*** *(p. 1737).*
- **dds::core::StringSeq disabled_metrics_selection** () const

  Getter (see setter with the same name)

8.187.1 Detailed Description

<<***extension***>> *(p. 354)* Configures event and periodic metrics collection and distribution for a specific set of observable resources.

8.187.2 Constructor & Destructor Documentation

8.187.2.1 MonitoringMetricSelection()

```
rti::core::MonitoringMetricSelection::MonitoringMetricSelection ( )  [default]
```

Creates an instance with the default settings.

8.187.3 Member Function Documentation

8.187.3.1 resource_selection() [1/2]

```
MonitoringMetricSelection & rti::core::MonitoringMetricSelection::resource_selection (
            const std::string & the_resource_selection )
```

An expression pattern that selects a subset of resources by matching the pattern to the resource names. **rti::core::MonitoringMetricSelection::enabled_metrics_selection** (p. 1738) and **rti::core::MonitoringMetricSelection::disabled_metrics_selection** (p. 1739) are applied to this subset of resources.

Parameters

the_resource_selection	A string.

Examples of resource expression patterns:

- /applications/myApp/domain_participants/myParticipant
- /applications/∗/domain_participants/∗/subscribers/∗
- /applications/myApp/domain_participants/GUID(1234.5678.4321.8765)
- //myEntity

The first expression refers to a DomainParticipant named "myParticipant" that belongs to an application named "myApp". The second expression applies to all the Subscribers in the system (resource name wildcards follow the POSIX fnmatch syntax). The third expression refers to a DomainParticipant with a specific resource GUID in the "myApp" application. The last expression uses the XPath "//" operator. It matches observable resources named "myEntity" no matter where they are located in the resource hiearchy.

See the Telemetry Data / Resources chapter of the RTI Connext Observability Framework documentation for further information on the observable resource names and expression patterns.

Categories

Mutable (p. 280), **Debuggability** (p. 172)

8.187.3.2 resource_selection() [2/2]

```
std::string rti::core::MonitoringMetricSelection::resource_selection ( ) const
```

Getter (see setter with the same name).

8.187.3.3 enabled_metrics_selection() [1/2]

```
MonitoringMetricSelection & rti::core::MonitoringMetricSelection::enabled_metrics_selection (
            const  dds::core::StringSeq & the_enabled_metrics_selection )
```

Sets A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by **rti::core::MonitoringMetricSelection::resource_selection** (p. 1737).

This sequence is evaluated first, followed by **rti::core::MonitoringMetricSelection::disabled_metrics_selection** (p. 1739). Therefore, if the same metric is enabled and disabled at the same time, the disablement will be the final result.

The patterns in the sequence are evaluated in order.

Examples of valid patterns:

- dds_data_writer_qos_durability_writer_depth
- dds_data_reader_qos_reliability_*
- dds_application_*

The first pattern refers to a specific DataWriter metric (**dds::core::policy::Durability::writer_depth** (p. 1439)). The second pattern refers to all the DataReader **dds::core::policy::Reliability** (p. 2149) metrics. The last pattern selects all the available application metrics.

See the Telemetry Data / Metrics chapter of the RTI Connext Observability Framework documentation for further information on the metric names and metric patterns.

Categories

Mutable (p. 280), **Debuggability** (p. 172)

8.187.3.4 enabled_metrics_selection() [2/2]

```
dds::core::StringSeq rti::core::MonitoringMetricSelection::enabled_metrics_selection ( ) const
```

Getter (see setter with the same name)

8.187.3.5 disabled_metrics_selection() [1/2]

```
MonitoringMetricSelection & rti::core::MonitoringMetricSelection::disabled_metrics_selection (
            const  dds::core::StringSeq & the_disabled_metrics_selection )
```

Sets A sequence of POSIX fnmatch patterns that match the names of the metrics that should not be collected and distributed for the observable resources selected by **rti::core::MonitoringMetricSelection::resource_selection** (p. 1737).

This sequence is evaluated after **rti::core::MonitoringMetricSelection::enabled_metrics_selection** (p. 1738). Therefore, if the same metric is enabled and disabled at the same time, the disablement will be the final result.

The patterns in the sequence are evaluated in order.

Examples of valid patterns:

- dds_data_writer_qos_durability_writer_depth
- dds_data_reader_qos_reliability_*
- dds_application_*

The first pattern refers to a specific DataWriter metric (**dds::core::policy::Durability::writer_depth** (p. 1439)). The second pattern refers to all the DataReader **dds::core::policy::Reliability** (p. 2149) metrics. The last pattern selects all the available application metrics.

See the Telemetry Data / Metrics chapter of the RTI Connext Observability Framework documentation for further information on the metric names and metric patterns.

Categories

Mutable (p. 280), **Debuggability** (p. 172)

8.187.3.6 disabled_metrics_selection() [2/2]

```
dds::core::StringSeq rti::core::MonitoringMetricSelection::disabled_metrics_selection ( ) const
```

Getter (see setter with the same name)

8.188 rti::core::MonitoringPeriodicDistributionSettings Class Reference

<<***extension***>> *(p. 354)* Configures the distribution of periodic metrics.

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **MonitoringPeriodicDistributionSettings** ()

  Creates an instance with the default settings.
- **MonitoringPeriodicDistributionSettings** & **datawriter_qos_profile_name** (const **rti::core::optional_value**< std::string > &the_datawriter_qos_profile_name)

  *Sets The fully qualified name of the profile used to configure the **dds::pub::DataWriter*** (p. *1138) that distributes periodic metrics.*
- **MonitoringPeriodicDistributionSettings** & **datawriter_qos_profile_name** (const char *the_datawriter_qos↩ _profile_name)

  *Sets The fully qualified name of the profile used to configure the **dds::pub::DataWriter*** (p. *1138) that distributes periodic metrics.*
- **rti::core::optional_value**< std::string > **datawriter_qos_profile_name** () const

  Getter (see setter with the same name).
- **MonitoringPeriodicDistributionSettings** & **thread** (const **rti::core::ThreadSettings** &the_thread)

  *<<**extension**>> (p. 116) The settings of the periodic metric thread.*
- const **rti::core::ThreadSettings** & **thread** () const

  Getter (see setter with the same name).
- **rti::core::ThreadSettings** & **thread** ()

  Getter (see setter with the same name).
- **MonitoringPeriodicDistributionSettings** & **polling_period** (const **dds::core::Duration** &the_polling_period)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) Period at which the periodic metric thread polls and publishes the periodic metrics.*
- **dds::core::Duration polling_period** () const

  Getter (see setter with the same name)

8.188.1 Detailed Description

*<<**extension**>> (p. 354)* Configures the distribution of periodic metrics.

Periodic metrics change often, and they are polled and published periodically by a thread created by RTI Monitoring Library 2.0.

RTI Monitoring Library 2.0 obtains periodic metrics by polling the current value of periodic statuses such as **rti::core↩ ::status::DataWriterProtocolStatus** (p. 1219).

The periodic metrics that will be distributed for an observable resource can be configured with **rti::core::Monitoring↩ TelemetryData::metrics** (p. 1745).

8.188.2 Constructor & Destructor Documentation

8.188.2.1 MonitoringPeriodicDistributionSettings()

```
rti::core::MonitoringPeriodicDistributionSettings::MonitoringPeriodicDistributionSettings ( )
[inline]
```

Creates an instance with the default settings.

8.188.3 Member Function Documentation

8.188.3.1 datawriter_qos_profile_name() [1/3]

```
MonitoringPeriodicDistributionSettings & rti::core::MonitoringPeriodicDistributionSettings↩
::datawriter_qos_profile_name (
            const  rti::core::optional_value< std::string > & the_datawriter_qos_profile_name )
```

Sets The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes periodic metrics.

Parameters

the_datawriter_qos_profile_name	An optional string. An unset value indicates that the RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_monitoring2()** (p. 479).

[default] Unset

The **dds::pub::DataWriter** (p. 1138) Topic is MONITORING_PERIODIC_TOPIC_NAME.

If null (the default value), then RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_↩ monitoring2()** (p. 479).

[default] null

Categories

Mutable (p. 280), **Debuggability** (p. 172), **QoS Profile Management** (p. 220)

8.188.3.2 datawriter_qos_profile_name() [2/3]

```
MonitoringPeriodicDistributionSettings & rti::core::MonitoringPeriodicDistributionSettings↩
::datawriter_qos_profile_name (
            const char * the_datawriter_qos_profile_name )
```

Sets The fully qualified name of the profile used to configure the **dds::pub::DataWriter** (p. 1138) that distributes periodic metrics.

Parameters

the_datawriter_qos_profile_name	The profile name. Cannot be NULL.

The **dds::pub::DataWriter** (p. 1138) Topic is MONITORING_PERIODIC_TOPIC_NAME.

If null (the default value), then RTI Monitoring Library 2.0 uses **rti::core::builtin_profiles::qos_lib::generic_↩ monitoring2()** (p. 479).

[default] null

Categories

Mutable (p. 280), **Debuggability** (p. 172), **QoS Profile Management** (p. 220)

8.188.3.3 datawriter_qos_profile_name() [3/3]

```
rti::core::optional_value< std::string > rti::core::MonitoringPeriodicDistributionSettings↩
::datawriter_qos_profile_name ( ) const
```

Getter (see setter with the same name).

8.188.3.4 thread() [1/3]

```
MonitoringPeriodicDistributionSettings & rti::core::MonitoringPeriodicDistributionSettings↩
::thread (
            const  rti::core::ThreadSettings & the_thread )
```

<<***extension***>> *(*p. *116)* The settings of the periodic metric thread.

The periodic metric thread periodically polls and publishes periodic event metrics.

The thread runs at the period configured using **rti::core::MonitoringPeriodicDistributionSettings::polling_period** (p. 1744).

[default] DDS_THREAD_SETTINGS_DEFAULT

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Concurrency** (p. 164)

8.188.3.5 thread() [2/3]

```
const  rti::core::ThreadSettings & rti::core::MonitoringPeriodicDistributionSettings::thread ( )
const
```

Getter (see setter with the same name).

8.188.3.6 thread() [3/3]

```
rti::core::ThreadSettings & rti::core::MonitoringPeriodicDistributionSettings::thread ( )
```

Getter (see setter with the same name).

8.188.3.7 polling_period() [1/2]

```
MonitoringPeriodicDistributionSettings & rti::core::MonitoringPeriodicDistributionSettings↩
::polling_period (
            const  dds::core::Duration & the_polling_period )
```

<<***basic***>> (p. *104)* <<***extension***>> (p. *116)* Period at which the periodic metric thread polls and publishes the periodic metrics.

[default] 5 seconds

[range] [1 nanosecond, 4294967 seconds (50 days)]

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Data Availability** (p. 167)

8.188.3.8 polling_period() [2/2]

```
dds::core::Duration rti::core::MonitoringPeriodicDistributionSettings::polling_period ( ) const
```

Getter (see setter with the same name)

8.189 rti::core::MonitoringTelemetryData Class Reference

<<***extension***>> (p. *354)* Configures the telemetry data that will be distributed.

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **MonitoringTelemetryData** & **metrics** (const **MonitoringMetricSelectionSeq** &the_metrics)

  *Sets the sequence of **rti::core::MonitoringMetricSelection** (p. 1736) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.*
- **MonitoringMetricSelectionSeq** **metrics** () const

  Gets the metrics (see setter).
- **MonitoringTelemetryData** & **logs** (const **MonitoringLoggingForwardingSettings** &the_logs)

  *Sets the **rti::core::MonitoringLoggingForwardingSettings** (p. 1733) containing the **rti::config::SyslogVerbosity** (p. 458) levels that will be forwarded for the different **rti::config::LogFacility** (p. 459).*
- const **MonitoringLoggingForwardingSettings** & **logs** () const

  Gets the metrics by const-reference (see setter).
- **MonitoringLoggingForwardingSettings** & **logs** ()

  Gets the metrics by reference (see setter).

8.189.1 Detailed Description

<<***extension***>> *(p. 354)* Configures the telemetry data that will be distributed.

8.189.2 Member Function Documentation

8.189.2.1 metrics() [1/2]

```
MonitoringTelemetryData & rti::core::MonitoringTelemetryData::metrics (
            const  MonitoringMetricSelectionSeq & the_metrics )
```

Sets the sequence of **rti::core::MonitoringMetricSelection** (p. 1736) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.

Parameters

the_metrics	sequence of **rti::core::MonitoringMetricSelection** (p. 1736).

The different **rti::core::MonitoringMetricSelection** (p. 1736) in the sequence are evaluated in order.

[default] An empty sequence, meaning that no metrics will be collected and distributed for any observable resource.

See also

rti::core::MonitoringEventDistributionSettings (p. 1722) and **rti::core::MonitoringPeriodicDistribution↩ Settings** (p. 1740) for further information on how RTI Monitoring Library 2.0 distributes **metrics** (p. 1745).

Categories

Mutable (p. 280), **Debuggability** (p. 172)

8.189.2.2 metrics() [2/2]

```
MonitoringMetricSelectionSeq rti::core::MonitoringTelemetryData::metrics ( ) const
```

Gets the metrics (see setter).

8.189.2.3 logs() [1/3]

```
MonitoringTelemetryData & rti::core::MonitoringTelemetryData::logs (
            const  MonitoringLoggingForwardingSettings & the_logs )
```

Sets the **rti::core::MonitoringLoggingForwardingSettings** (p. 1733) containing the **rti::config::SyslogVerbosity** (p. 458) levels that will be forwarded for the different **rti::config::LogFacility** (p. 459).

Parameters

the_logs	**rti::core::MonitoringLoggingForwardingSettings** (p. 1733) object.

See also

rti::core::MonitoringLoggingDistributionSettings (p. 1727) for further information on how RTI Monitoring Library 2.0 distributes log messages.

Categories

Mutable (p. 280), **Debuggability** (p. 172), **Logging** (p. 201)

8.189.2.4 logs() [2/3]

```
const  MonitoringLoggingForwardingSettings & rti::core::MonitoringTelemetryData::logs ( ) const
```

Gets the metrics by const-reference (see setter).

8.189.2.5 logs() [3/3]

```
MonitoringLoggingForwardingSettings & rti::core::MonitoringTelemetryData::logs ( )
```

Gets the metrics by reference (see setter).

8.190 rti::core::policy::MulticastMapping Class Reference

<<***extension***>> *(*p. *116)* Type representing a list of multicast mapping elements.

```
#include <CorePolicy.hpp>
```

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **MulticastMapping** & **addresses** (const std::string &the_addresses)

  <<***extension***>> *(*p. *116) A string containing a comma-separated list of IP addresses or IP address ranges to be used to receive multicast traffic for the entity with a topic that matches the* ***rti::core::policy::MulticastMapping::topic_expression*** *(*p. *1748).*
- **rti::core::optional_value**< std::string > **addresses** () const

  Getter (see setter with the same name)
- **MulticastMapping** & **topic_expression** (const std::string &the_topic_expression)

  <<***extension***>> *(*p. *116) A regular expression that will be used to map topic names to corresponding multicast receive addresses.*
- **rti::core::optional_value**< std::string > **topic_expression** () const

  Getter (see setter with the same name)
- **MulticastMapping** & **mapping_function** (const TransportMulticastMappingFunction &the_mappings)

  <<***extension***>> *(*p. *116) Specifies a function that will define the mapping between a topic name and a specific multicast address from a list of addresses.*
- const TransportMulticastMappingFunction & **mapping_function** () const

  Getter (see setter with the same name)

8.190.1 Detailed Description

<<***extension***>> *(*p. *116)* Type representing a list of multicast mapping elements.

A multicast mapping element specifies a string containing a list of IP addresses, a topic expression and a mapping function.

QoS:

rti::core::policy::TransportMulticastMapping (p. 2547)

8.190.2 Member Function Documentation

8.190.2.1 addresses() [1/2]

```
MulticastMapping & rti::core::policy::MulticastMapping::addresses (
            const std::string & the_addresses )
```

<<***extension***>> *(*p. *116)* A string containing a comma-separated list of IP addresses or IP address ranges to be used to receive *multicast* traffic for the entity with a topic that matches the **rti::core::policy::MulticastMapping::topic_↩ expression** (p. 1748).

The string must contain IPv4 or IPv6 addresses separated by commas. For example: "239.255.100.1,239.255.100.↩ 2,239.255.100.3"

You may specify ranges of addresses by enclosing the start address and the end address in square brackets. For example: "[239.255.100.1,239.255.100.3]"

You may combine the two approaches. For example:

"239.255.200.1,[239.255.100.1,239.255.100.3], 239.255.200.3"

IPv4 addresses must be specified in Dot-decimal notation.

IPv6 addresses must be specified using 8 groups of 16-bit hexadecimal values separated by colons. For example: FF00:0000:0000:0000:0202:B3FF:FE1E:8329

Leading zeroes can be skipped. For example: "FF00:0:0:0:202:B3FF:FE1E:8329"

You may replace a consecutive number of zeroes with a double colon, but only once within an address. For example: "FF00::202:B3FF:FE1E:8329"

[default] NULL

8.190.2.2 addresses() [2/2]

```
rti::core::optional_value< std::string > rti::core::policy::MulticastMapping::addresses ( )
const
```

Getter (see setter with the same name)

8.190.2.3 topic_expression() [1/2]

```
MulticastMapping & rti::core::policy::MulticastMapping::topic_expression (
            const std::string & the_topic_expression )
```

<<***extension***>> *(*p. *116)* A regular expression that will be used to map topic names to corresponding multicast receive addresses.

A topic name must match the expression before a corresponding address is assigned.

[default] NULL

8.190.2.4 topic_expression() [2/2]

```
rti::core::optional_value< std::string > rti::core::policy::MulticastMapping::topic_expression (
) const
```

Getter (see setter with the same name)

8.190.2.5 mapping_function() [1/2]

```
MulticastMapping & rti::core::policy::MulticastMapping::mapping_function (
            const TransportMulticastMappingFunction & the_mappings )
```

<<***extension***>> *(*p. *116)* Specifies a function that will define the mapping between a topic name and a specific multicast address from a list of addresses.

This function is optional. If not specified, the middleware will use a hash function to perform the mapping.

8.190.2.6 mapping_function() [2/2]

```
const TransportMulticastMappingFunction & rti::core::policy::MulticastMapping::mapping_function (
) const
```

Getter (see setter with the same name)

8.191 rti::core::policy::MultiChannel Class Reference

<<***extension***>> *(*p. *116)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **MultiChannel** ()

  Creates the default policy.
- **MultiChannel** (const **rti::core::ChannelSettingsSeq** &the_channels, const std::string &the_filter_name= **rti::topic::stringmatch_filter_name**())

  Creates an instance with the specified channels and filter name.
- **MultiChannel** & **channels** (const **rti::core::ChannelSettingsSeq** &the_channels)

  *<<**extension**>> (p. 116) A sequence of **rti::core::ChannelSettings** (p. 921) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.*
- **rti::core::ChannelSettingsSeq channels** () const

  Getter (see the setter with the same name)
- **MultiChannel** & **filter_name** (const std::string &the_filter_name)

  *<<**extension**>> (p. 116) Name of the filter class used to describe the filter expressions of a **MultiChannel** (p. 1749) DataWriter.*
- std::string **filter_name** () const

  Getter (see the setter with the same name)

8.191.1 Detailed Description

*<<**extension**>> (p. 116)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

This QoS policy is used to partition the data published by a **dds::pub::DataWriter** (p. 1138) across multiple channels. A *channel* is defined by a filter expression and a sequence of multicast locators.

Entity:

dds::pub::DataWriter (p. 1138)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.191.2 Usage

By using this QoS, a **dds::pub::DataWriter** (p. 1138) can be configured to send data to different multicast groups based on the content of the data. Using syntax similar to those used in Content-Based Filters, you can associate different multicast addresses with filter expressions that operate on the values of the fields within the data. When your application's code calls **dds::pub::DataWriter::write()** (p. 1148), data is sent to any multicast address for which the data passes the filter.

Multi-channel DataWriters can be used to trade off network bandwidth with the unnecessary processing of unwanted data for situations where there are multiple DataReaders that are interested in different subsets of data that come from the same data stream (Topic). For example, in Financial applications, the data stream may be quotes for different stocks at an exchange. Applications usually only want to receive data (quotes) for only a subset of the stocks being traded.

In tracking applications, a data stream may carry information on hundreds or thousands of objects being tracked, but again, applications may only be interested in a subset.

The problem is that the most efficient way to deliver data to multiple applications is to use multicast, so that a data value is only sent once on the network for any number of subscribers to the data. However, using multicast, an application will receive *all* of the data sent and not just the data in which it is interested, thus extra CPU time is wasted to throw away unwanted data. With this QoS, you can analyze the data-usage patterns of your applications and optimize network vs. CPU usage by partitioning the data into multiple multicast streams. While network bandwidth is still being conserved by sending data only once using multicast, most applications will only need to listen to a subset of the multicast addresses and receive a reduced amount of unwanted data.

Your system can gain more of the benefits of using multiple multicast groups if your network uses Layer 2 Ethernet switches. Layer 2 switches can be configured to only route multicast packets to those ports that have added membership to specific multicast groups. Using those switches will ensure that only the multicast packets used by applications on a node are routed to the node; all others are filtered-out by the switch.

8.191.3 Constructor & Destructor Documentation

8.191.3.1 MultiChannel() [1/2]

```
rti::core::policy::MultiChannel::MultiChannel ( )  [inline]
```

Creates the default policy.

8.191.3.2 MultiChannel() [2/2]

```
rti::core::policy::MultiChannel::MultiChannel (
            const  rti::core::ChannelSettingsSeq & the_channels,
            const std::string & the_filter_name =  rti::topic::stringmatch_filter_name() )  [inline]
```

Creates an instance with the specified channels and filter name.

8.191.4 Member Function Documentation

8.191.4.1 channels() [1/2]

```
MultiChannel & rti::core::policy::MultiChannel::channels (
            const  rti::core::ChannelSettingsSeq & the_channels )
```

<<***extension***>> *(*p. *116)* A sequence of **rti::core::ChannelSettings** (p. 921) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.

A sequence length of zero indicates the **rti::core::policy::MultiChannel** (p. 1749) is not in use.

The sequence length cannot be greater than **rti::core::policy::DomainParticipantResourceLimits::channel_seq_↩ max_length** (p. 1423).

[default] Empty sequence.

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Data Availability** (p. 167)

8.191.4.2 channels() [2/2]

```
rti::core::ChannelSettingsSeq rti::core::policy::MultiChannel::channels ( ) const
```

Getter (see the setter with the same name)

8.191.4.3 filter_name() [1/2]

```
MultiChannel & rti::core::policy::MultiChannel::filter_name (
            const std::string & the_filter_name )
```

<<***extension***>> *(*p. *116)* Name of the filter class used to describe the filter expressions of a **MultiChannel** (p. 1749) DataWriter.

The following builtin filters are supported:

- **rti::topic::sql_filter_name** (p. 49)
- **rti::topic::stringmatch_filter_name** (p. 49)

[default] rti::topic::stringmatch_filter_name (p. 49)

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Data Availability** (p. 167), **Content Filtering** (p. 165)

8.191.4.4 filter_name() [2/2]

```
std::string rti::core::policy::MultiChannel::filter_name ( ) const
```

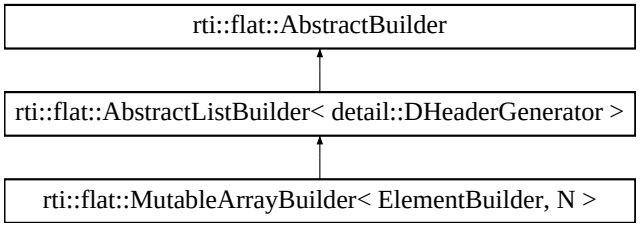
Getter (see the setter with the same name)

8.192 rti::flat::MutableArrayBuilder< ElementBuilder, N > Class Template Reference

Builds an array member of variable-size elements.

```
#include <SequenceBuilders.hpp>
```

Inheritance diagram for rti::flat::MutableArrayBuilder< ElementBuilder, N >:



Public Types

- typedef **MutableArrayOffset**< typename ElementBuilder::Offset, N > **Offset**
  The related Offset type.

Public Member Functions

- ElementBuilder **build_next** ()
  Begins building the next element.
- **Offset finish** ()
  Finishes building the array.

Additional Inherited Members

8.192.1 Detailed Description

template<typename ElementBuilder, unsigned int N>
class rti::flat::MutableArrayBuilder< ElementBuilder, N >

Builds an array member of variable-size elements.

Template Parameters

ElementBuilder	The Builder type for the elements of the array
N	The array bound; the exact number of elements this array builder must build. For multidimensional arrays, N is the product of each dimension bound.

Each element of this array needs to be built using the ElementBuilder returned by **build_next()** (p. 1754). N elements exactly must be built. Unlike a sequence Builder, it is not possible to finish an array with less than N elements.

This example shows how to use a **MutableArrayBuilder** (p. 1753) to build an array member of **MyFlatMutableBuilder** (p. 1763):

```
MyFlatMutableBuilder builder = rti::flat::build_data(writer);
auto array_builder = builder.build_my_mutable_array();
for (int i = 0; i < 10; i++) {
    FlatMutableBarBuilder element_builder = array_builder.build_next();
    // ... build element
    element_builder.finish();
}
array_builder.finish();
```

Note that Builder types are not necessary for arrays of fixed-size elements, since they are added at once (see **MyFlat↩ MutableBuilder::add_my_final_array()** (p. 1768)).

8.192.2 Member Typedef Documentation

8.192.2.1 Offset

```
template<typename ElementBuilder , unsigned int N>
typedef MutableArrayOffset<typename ElementBuilder::Offset, N> rti::flat::MutableArrayBuilder<
ElementBuilder, N >::Offset
```

The related Offset type.

8.192.3 Member Function Documentation

8.192.3.1 build_next()

```
template<typename ElementBuilder , unsigned int N>
ElementBuilder rti::flat::MutableArrayBuilder< ElementBuilder, N >::build_next ( ) [inline]
```

Begins building the next element.

Before calling **build_next()** (p. 1754) to create a new element, the element Builder returned by a previous call to build↩ _next must have been finished.

References **rti::flat::AbstractListBuilder< detail::DHeaderGenerator >::element_count()**.

8.192.3.2 finish()

```
template<typename ElementBuilder , unsigned int N>
Offset  rti::flat::MutableArrayBuilder< ElementBuilder, N >::finish ( )  [inline]
```

Finishes building the array.

Precondition

build_next() (p. 1754) must have been succesfully called N times exactly.

Returns

The Offset to the member that has been built.

See also

discard() (p. 784)

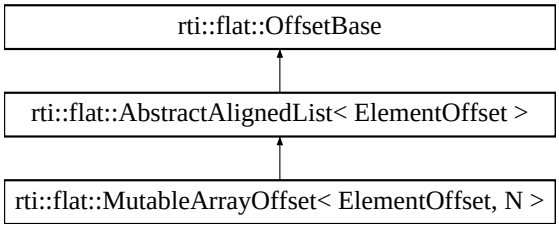
References **rti::flat::AbstractListBuilder< detail::DHeaderGenerator >::element_count()**.

8.193 rti::flat::MutableArrayOffset< ElementOffset, N > Class Template Reference

Offset to an array of variable-size elements.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::MutableArrayOffset< ElementOffset, N >:



Public Member Functions

- ElementOffset **get_element** (unsigned int i)

  Gets the Offset to an element.

Additional Inherited Members

8.193.1 Detailed Description

template<typename ElementOffset, unsigned int N>
class rti::flat::MutableArrayOffset< ElementOffset, N >

Offset to an array of variable-size elements.

Template Parameters

ElementOffset	An Offset for a type of variable size, such as a mutable struct (**MyFlatMutableOffset** (p. 1770)), union, or **StringOffset** (p. 2394).
N	The array bound. For multidimensional arrays, `N` is the product of each dimension bound.

Represents an Offset to an array member and allows getting an Offset to each of its elements.

See also

FinalArrayOffset (p. 1567) encapsulates arrays of fixed-size elements

8.193.2 Member Function Documentation

8.193.2.1 get_element()

```
template<typename ElementOffset , unsigned int N>
ElementOffset  rti::flat::MutableArrayOffset< ElementOffset, N >::get_element (
             unsigned int i )  [inline]
```

Gets the Offset to an element.

Parameters

i	The zero-based index to the element

Returns

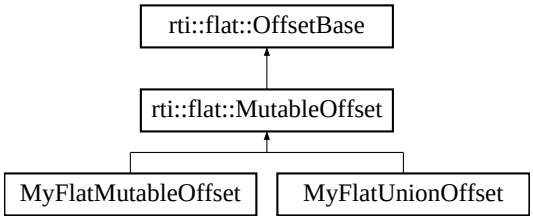
The Offset to the element in the i-th position

8.194 rti::flat::MutableOffset Class Reference

The base class of all Offsets to a final struct type.

`#include <Offset.hpp>`

Inheritance diagram for rti::flat::MutableOffset:



Additional Inherited Members

8.194.1 Detailed Description

The base class of all Offsets to a final struct type.

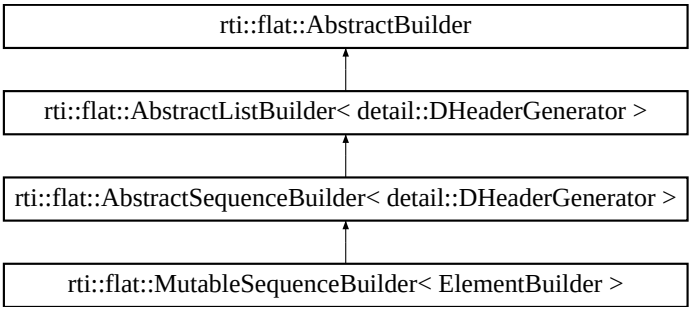
This class contains only implementation details; all the public accessors are defined in the generated type (**MyFlat↩ MutableOffset** (p. 1770)).

8.195 rti::flat::MutableSequenceBuilder< ElementBuilder > Class Template Reference

Builds a sequence member of variable-size elements.

`#include <SequenceBuilders.hpp>`

Inheritance diagram for rti::flat::MutableSequenceBuilder< ElementBuilder >:



Public Types

- typedef **SequenceOffset**< typename ElementBuilder::Offset > **Offset**
  The related Offset type.

Public Member Functions

- ElementBuilder **build_next** ()
  Begins building the next element.
- **Offset finish** ()
  Finishes building the sequence.

Additional Inherited Members

8.195.1 Detailed Description

template<typename ElementBuilder>
class rti::flat::MutableSequenceBuilder< ElementBuilder >

Builds a sequence member of variable-size elements.

Template Parameters

ElementBuilder	The Builder type for the elements of the sequence

To build the elements use the ElementBuilder returned by **build_next()** (p. 1758). An empty sequence can be built by calling **finish()** (p. 1758) without any call to **build_next()** (p. 1758).

This class doesn't enforce the sequence bound set in IDL.

The following example uses a **MutableSequenceBuilder** (p. 1757) to initialize a sequence member of **MyFlatMutable↩ Builder** (p. 1763) with two elements:

```
MyFlatMutableBuilder builder = rti::flat::build_data(writer);
auto seq_builder = builder.build_my_mutable_seq();
auto element_builder = seq_builder.build_next();
// ... build the first element
element_builder.finish();
element_builder = seq_builder.build_next();
// ... build the second element
element_builder.finish();

seq_builder.finish();
```

8.195.2 Member Typedef Documentation

8.195.2.1 Offset

```
template<typename ElementBuilder >
typedef  SequenceOffset<typename ElementBuilder::Offset>  rti::flat::MutableSequenceBuilder<
ElementBuilder >::Offset
```

The related Offset type.

8.195.3 Member Function Documentation

8.195.3.1 build_next()

```
template<typename ElementBuilder >
ElementBuilder  rti::flat::MutableSequenceBuilder< ElementBuilder >::build_next ( )  [inline]
```

Begins building the next element.

Before calling **build_next()** (p. 1758) to create a new element, the application must have called **finish()** (p. 1758) on the previous element Builder.

8.195.3.2 finish()

```
template<typename ElementBuilder >
Offset rti::flat::MutableSequenceBuilder< ElementBuilder >::finish ( ) [inline]
```

Finishes building the sequence.

Returns

An Offset to the member that has been built.

See also

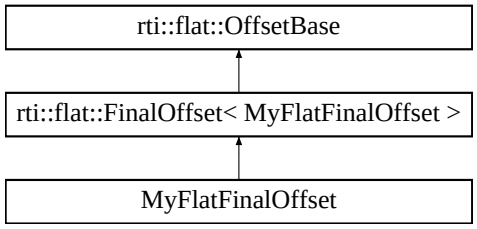
discard() (p. 784)

8.196 MyFlatFinalOffset Class Reference

Represents the Offset to an arbitrary user-defined FlatData final IDL struct.

```
#include <Offset.hpp>
```

Inheritance diagram for MyFlatFinalOffset:



Public Types

- typedef MyFlatFinalConstOffset **ConstOffset**

  The equivalent read-only Offset type.

Public Member Functions

- **MyFlatFinalOffset** ()

  Creates a null Offset.
- int32_t **my_primitive** () const

  Retrieves the value for a primitive member.
- FlatFinalBar::ConstOffset **my_complex** () const

  Retrieves a const Offset to a complex member.
- const **rti::flat::PrimitiveArrayOffset**< int32_t, 10 > **my_primitive_array** () const

  Retrieves a const Offset to a primitive array.
- **rti::flat::FinalArrayOffset**< FlatFinalBar::ConstOffset, 10 > **my_complex_array** () const

  Retrieves a const Offset to a complex array.
- bool **my_primitive** (int32_t value)

  Sets the value for a primitive member.
- FlatFinalBar::Offset **my_complex** ()

  Retrieves a non-const Offset to a complex member.
- **rti::flat::PrimitiveArrayOffset**< int32_t, 10 > **my_primitive_array** ()

  Retrieves a non-const Offset to a primitive array.
- **rti::flat::FinalArrayOffset**< FlatFinalBar::Offset, 10 > **my_complex_array** ()

  Retrieves a non-const Offset to a complex array.

8.196.1 Detailed Description

Represents the Offset to an arbitrary user-defined FlatData final IDL struct.

This example type represents the Offset type that **rtiddsgen** would generate for **MyFlatFinal** (p. 413).

It provides accessors for each of its members. Accessors can return other Offsets or primitive values.

An Offset to a final type may meet the requirements to be cast to its equivalent plain C++ type (see **rti::flat::plain_cast()** (p. 417)).

8.196.2 Member Typedef Documentation

8.196.2.1 ConstOffset

`typedef MyFlatFinalConstOffset` **`MyFlatFinalOffset::ConstOffset`**

The equivalent read-only Offset type.

Each Offset for a user type has an equivalent const Offset that doesn't provide the methods to modify the Sample. For example, when reading data from a **dds::sub::DataReader** (p. 976), a const Sample's Sample::root() function returns a const Offset.

8.196.3 Constructor & Destructor Documentation

8.196.3.1 MyFlatFinalOffset()

```
MyFlatFinalOffset::MyFlatFinalOffset ( )  [inline]
```

Creates a null Offset.

Postcondition

is_null() (p. 1876)

8.196.4 Member Function Documentation

8.196.4.1 my_primitive() [1/2]

```
int32_t MyFlatFinalOffset::my_primitive ( ) const
```

Retrieves the value for a primitive member.

8.196.4.2 my_complex() [1/2]

```
FlatFinalBar::ConstOffset MyFlatFinalOffset::my_complex ( ) const
```

Retrieves a const Offset to a complex member.

FlatFinalBar is another arbitrary user-defined final FlatData type.

8.196.4.3 my_primitive_array() [1/2]

```
const  rti::flat::PrimitiveArrayOffset< int32_t, 10 > MyFlatFinalOffset::my_primitive_array ( )
const
```

Retrieves a const Offset to a primitive array.

8.196.4.4 my_complex_array() [1/2]

```
rti::flat::FinalArrayOffset< FlatFinalBar::ConstOffset, 10 > MyFlatFinalOffset::my_complex_array
( ) const
```

Retrieves a const Offset to a complex array.

8.196.4.5 my_primitive() [2/2]

```
bool MyFlatFinalOffset::my_primitive (
            int32_t value )
```

Sets the value for a primitive member.

8.196.4.6 my_complex() [2/2]

```
FlatFinalBar::Offset MyFlatFinalOffset::my_complex ( )
```

Retrieves a non-const Offset to a complex member.

FlatFinalBar is another arbitrary user-defined final FlatData type.

8.196.4.7 my_primitive_array() [2/2]

```
rti::flat::PrimitiveArrayOffset< int32_t, 10 > MyFlatFinalOffset::my_primitive_array ( )
```

Retrieves a non-const Offset to a primitive array.

8.196.4.8 my_complex_array() [2/2]

```
rti::flat::FinalArrayOffset< FlatFinalBar::Offset, 10 > MyFlatFinalOffset::my_complex_array ( )
```

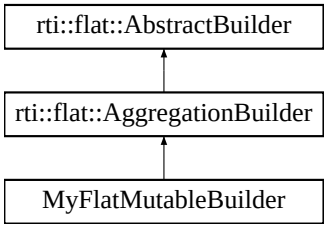
Retrieves a non-const Offset to a complex array.

8.197 MyFlatMutableBuilder Class Reference

Represents the Builder for an arbitrary user-defined mutable type.

`#include <Builder.hpp>`

Inheritance diagram for MyFlatMutableBuilder:



Public Types

- typedef **MyFlatMutableOffset Offset**
  The related offset type.

Public Member Functions

- **MyFlatMutableBuilder** ()
  Creates an invalid Builder.
- **MyFlatMutableBuilder** (unsigned char ∗buffer, int32_t size, bool initialize_members=false)
  Construct a Builder with an arbitrary buffer.
- **Offset finish** ()
  Finishes building a member.
- **MyFlatMutable** ∗ **finish_sample** ()
  Finishes building a sample.
- bool **add_my_primitive** (int32_t value)
  Adds a primitive member.
- bool **add_my_optional_primitive** (int32_t value)
  Adds a primitive member.
- **rti::flat::PrimitiveArrayOffset**< int32_t, 10 > **add_my_primitive_array** ()
  Adds a primitive array member.
- **rti::flat::PrimitiveSequenceBuilder**< int32_t > **build_my_primitive_seq** ()
  Begins building a primitive-sequence member.
- **MyFlatFinalOffset add_my_final** ()
  Adds a final complex member.
- **rti::flat::FinalAlignedArrayOffset**< **MyFlatFinalOffset**, 10 > **add_my_final_array** ()
  Adds an array member of final complex elements.
- **rti::flat::FinalSequenceBuilder**< **MyFlatFinalOffset** > **build_my_final_seq** ()
  Begins building a sequence member of final complex elements.
- FlatMutableBarBuilder **build_my_mutable** ()

Begins building a mutable complex member.

- **rti::flat::MutableArrayBuilder**< FlatMutableBarBuilder, 10 > **build_my_mutable_array** ()

  Begins building an array member of mutable complex elements.
- **rti::flat::MutableSequenceBuilder**< FlatMutableBarBuilder > **build_my_mutable_seq** ()

  Begins building a sequence member of mutable complex elements.
- **rti::flat::StringBuilder build_my_string** ()

  Begins building a string member.
- **rti::flat::MutableSequenceBuilder**< **rti::flat::StringBuilder** > **build_my_string_seq** ()

  Begins building a sequence member of string elements.

Additional Inherited Members

8.197.1 Detailed Description

Represents the Builder for an arbitrary user-defined mutable type.

This example type represents the **Builder** (p. 409) type that **rtiddsgen** would generate for **MyFlatMutable** (p. 413) and allows creating a sample or a member of that type.

The most common way to create a Builder is **rti::flat::build_data()** (p. 411), which obtains a managed sample from a **dds::pub::DataWriter** (p. 1138) as described in **Publishing FlatData** (p. 420). It is also possible to create a Builder with an arbitrary buffer using the constructor that receives the buffer and its size.

When a Builder is created, the type is empty–it doesn't contain any members. The Builder provides functions to create each member. There are two kind of functions: `add` functions and `build` functions.

8.197.1.1 Adding fixed-size members

Fixed-size members are "added", and the corresponding function is called **`add_<member_name>`**. "Add" functions directly place the member in the buffer, and return an **Offset** (p. 414) that allows setting its values.

By default, after calling `add_<member_name>` the member values are uninitialized. This behavior can be changed by creating the writer with **rti::core::policy::DataWriterResourceLimits::initialize_writer_loaned_sample** (p. 1248); this is however not recommended in general because of the performance impact for large types.

```
MyFlatMutableBuilder builder = ...;
MyFinalFooOffset member_offset = builder.add_my_final();
member_offset.my_primitive(10);
// ... add/build more members
```

8.197.1.2 Building variable-size members

Variable-size members are "built", and the function is called **`build_<member_name>`**. "Build" functions return another **Builder** (p. 409) to build the member.

```
MyFlatMutableBuilder builder = ...;
auto member_builder = builder.build_my_mutable();
// ... use member_builder
member_builder.finish();
// ... add/build more members
```

While a member is being built, the parent Builder is a **bound** state, which prevents any changes to it until the member has been finished by calling `member_builder.finish()`. In particular, the member Builder must be finished before adding or building any other member, or before finishing `builder`.

8.197.1.3 Choosing which members are included

The `add/build` function for each member can be called one or zero times. That is, all members are optional, even those without the `@optional` IDL annotation. However `@key` member must be added or **dds::pub::DataWriter::write()** (p. 1148) will fail.

FlatData samples received by **dds::sub::DataReader** (p. 976) will not contain the members that were not added/built– the corresponding member getters return a null Offset. A **dds::sub::DataReader** (p. 976) for an equivalent non-FlatData (plain) definition of this type will assign default values to any non-optional members that were not present in the sample when written.

It is not permitted to add a member more than once, but builders don't enforce this. Trying to build/add a member more than once may cause the Builder to overflow (see **Builder Error Management** (p. 410)). If it doesn't, it may be possible to write and read the data samples, but the duplicate members will be ignored.

See also

FlatData Builders (p. 409)

8.197.2 Member Typedef Documentation

8.197.2.1 Offset

```
typedef MyFlatMutableOffset MyFlatMutableBuilder::Offset
```

The related offset type.

8.197.3 Constructor & Destructor Documentation

8.197.3.1 MyFlatMutableBuilder() [1/2]

```
MyFlatMutableBuilder::MyFlatMutableBuilder ( ) [inline]
```

Creates an invalid Builder.

Postcondition

!is_valid()

Note

Top-level builders are created with **rti::flat::build_data()** (p. 411), and member builders are created with the corresponding `build_<member>` function.

8.197.3.2 MyFlatMutableBuilder() [2/2]

```
MyFlatMutableBuilder::MyFlatMutableBuilder (
            unsigned char * buffer,
            int32_t size,
            bool initialize_members = false )  [inline]
```

Construct a Builder with an arbitrary buffer.

Note

The recommended way to create a Builder is **rti::flat::build_data()** (p. 411) as described in **Publishing FlatData** (p. 420). This constructor provides an alternative option to use an arbitrary data buffer.

Parameters

buffer	The buffer that will be used to build the data sample
size	The size of the buffer
initialize_members	Whether fixed-size elements added to this Builder will be initialized to their default values or will be left uninitialized (more efficient).

If the sample being built overflows the buffer size, the add/build operations will fail. See **Builder Error Management** (p. 410).

8.197.4 Member Function Documentation

8.197.4.1 finish()

```
Offset MyFlatMutableBuilder::finish ( )
```

Finishes building a member.

Returns

The Offset to the member (normally it can be ignored)

Precondition

is_nested() (p. 784). That is, this object must be a member Builder, not a sample Builder.

8.197.4.2 finish_sample()

```
MyFlatMutable * MyFlatMutableBuilder::finish_sample ( )
```

Finishes building a sample.

Precondition

!is_nested() (p. 784). That is, this object must be a sample Builder, not a member Builder.

Postcondition

!is_valid() (p. 784). That is, this Builder can no longer be used.

Returns

The sample, ready to be written.

This function completes and returns the sample built with this Builder.

See also

rti::flat::build_data() (p. 411), the function to create a **dds::pub::DataWriter** (p. 1138)-managed sample Builder.

8.197.4.3 add_my_primitive()

```
bool MyFlatMutableBuilder::add_my_primitive (
            int32_t value )
```

Adds a primitive member.

8.197.4.4 add_my_optional_primitive()

```
bool MyFlatMutableBuilder::add_my_optional_primitive (
            int32_t value )
```

Adds a primitive member.

8.197.4.5 add_my_primitive_array()

```
rti::flat::PrimitiveArrayOffset< int32_t, 10 > MyFlatMutableBuilder::add_my_primitive_array ( )
```

Adds a primitive array member.

Returns

The Offset to the array, which can be used to set its values.

8.197.4.6 build_my_primitive_seq()

```
rti::flat::PrimitiveSequenceBuilder< int32_t > MyFlatMutableBuilder::build_my_primitive_seq ( )
```

Begins building a primitive-sequence member.

8.197.4.7 add_my_final()

```
MyFlatFinalOffset MyFlatMutableBuilder::add_my_final ( )
```

Adds a final complex member.

Returns

The Offset to the member, which can be used to set its values.

8.197.4.8 add_my_final_array()

```
rti::flat::FinalAlignedArrayOffset< MyFlatFinalOffset, 10 > MyFlatMutableBuilder::add_my_↩
final_array ( )
```

Adds an array member of final complex elements.

Returns

The Offset to the array, which can be used to set its values.

8.197.4.9 build_my_final_seq()

```
rti::flat::FinalSequenceBuilder< MyFlatFinalOffset > MyFlatMutableBuilder::build_my_final_seq (
)
```

Begins building a sequence member of final complex elements.

8.197.4.10 build_my_mutable()

```
FlatMutableBarBuilder MyFlatMutableBuilder::build_my_mutable ( )
```

Begins building a mutable complex member.

FlatMutableBar represents another arbitrary mutable FlatData type.

8.197.4.11 build_my_mutable_array()

```
rti::flat::MutableArrayBuilder< FlatMutableBarBuilder, 10 > MyFlatMutableBuilder::build_my_↩
mutable_array ( )
```

Begins building an array member of mutable complex elements.

8.197.4.12 build_my_mutable_seq()

```
rti::flat::MutableSequenceBuilder< FlatMutableBarBuilder > MyFlatMutableBuilder::build_my_↩
mutable_seq ( )
```

Begins building a sequence member of mutable complex elements.

8.197.4.13 build_my_string()

```
rti::flat::StringBuilder MyFlatMutableBuilder::build_my_string ( )
```

Begins building a string member.

8.197.4.14 build_my_string_seq()

rti::flat::MutableSequenceBuilder< **rti::flat::StringBuilder** > MyFlatMutableBuilder::build_my_↩
string_seq ( )

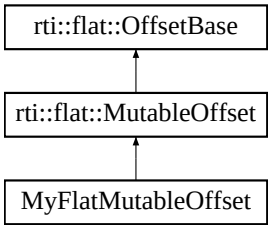
Begins building a sequence member of string elements.

8.198 MyFlatMutableOffset Class Reference

Represents the Offset to an arbitrary user-defined FlatData mutable IDL struct.

`#include <Offset.hpp>`

Inheritance diagram for MyFlatMutableOffset:



Public Types

- typedef MyFlatMutableConstOffset **ConstOffset**

  The equivalent read-only Offset type.

Public Member Functions

- **MyFlatMutableOffset** ()

  Creates a null Offset.
- int32_t **my_primitive** () const

  Retrieves the value for a primitive member.
- **rti::flat::PrimitiveConstOffset**< int32_t > **my_optional_primitive** () const

  Retrieves a const Offset to an optional primitive.
- const **rti::flat::PrimitiveArrayOffset**< int32_t, 10 > **my_primitive_array** () const

  Retrieves a const Offset to a primitive array.
- const **rti::flat::PrimitiveSequenceOffset**< int32_t > **my_primitive_seq** () const

  Retrieves a const Offset to a primitive sequence.
- **MyFlatFinal::ConstOffset** **my_final** () const

  Retrieves a const Offset to a complex member.
- **rti::flat::FinalAlignedArrayOffset**< **MyFlatFinal::ConstOffset**, 10 > **my_final_array** () const

  Retrieves a const Offset to a complex array.
- **rti::flat::SequenceOffset**< **MyFlatFinal::ConstOffset** > **my_final_seq** () const

Retrieves a const Offset to a complex sequence.

- FlatMutableBar::ConstOffset **my_mutable** () const

  Retrieves a const Offset to a complex member.
- **rti::flat::MutableArrayOffset**< FlatMutableBar::ConstOffset, 10 > **my_mutable_array** () const

  Retrieves a const Offset to a complex array.
- **rti::flat::SequenceOffset**< FlatMutableBar::ConstOffset > **my_mutable_seq** () const

  Retrieves a const Offset to a complex sequence.
- const **rti::flat::StringOffset** **my_string** () const

  Retrieves a const Offset to a string.
- **rti::flat::SequenceOffset**< const **rti::flat::StringOffset** > **my_string_seq** () const

  Retrieves a const Offset to a sequence of strings.
- bool **my_primitive** (int32_t value)

  Sets the value of a primitive member.
- **rti::flat::PrimitiveArrayOffset**< int32_t, 10 > **my_primitive_array** ()

  Retrieves a non-const Offset to a primitive array.
- **rti::flat::PrimitiveSequenceOffset**< int32_t > **my_primitive_seq** ()

  Retrieves a non-const Offset to a primitive sequence.
- **MyFlatFinal::Offset** **my_final** ()

  Retrieves a non-const Offset to a complex member.
- **rti::flat::FinalAlignedArrayOffset**< **MyFlatFinal::Offset**, 10 > **my_final_array** ()

  Retrieves a non-const Offset to a complex array.
- **rti::flat::SequenceOffset**< **MyFlatFinal::Offset** > **my_final_seq** ()

  Retrieves a non-const Offset to a complex sequence.
- FlatMutableBar::Offset **my_mutable** ()

  Retrieves a non-const Offset to a complex member.
- **rti::flat::MutableArrayOffset**< FlatMutableBar::Offset, 10 > **my_mutable_array** ()

  Retrieves a non-const Offset to a complex array.
- **rti::flat::SequenceOffset**< FlatMutableBar::Offset > **my_mutable_seq** ()

  Retrieves a non-const Offset to a complex sequence.
- **rti::flat::StringOffset** **my_string** ()

  Retrieves a non-const Offset to a string.
- **rti::flat::SequenceOffset**< **rti::flat::StringOffset** > **my_string_seq** ()

  Retrieves a non-const Offset to a sequence of strings.

8.198.1 Detailed Description

Represents the Offset to an arbitrary user-defined FlatData mutable IDL struct.

This example type represents the Offset type that **rtiddsgen** would generate for **MyFlatMutable** (p. 413).

It provides accessors for each of its members. Accessors can return other Offsets or primitive values.

8.198.2 Member Typedef Documentation

8.198.2.1 ConstOffset

```
typedef MyFlatMutableConstOffset  MyFlatMutableOffset::ConstOffset
```

The equivalent read-only Offset type.

Each Offset for a user type has an equivalent const Offset that doesn't allow modifying the underlying Sample. For example, when reading data from a **dds::sub::DataReader** (p. 976), a const Sample's Sample::root() function returns a const Offset.

8.198.3 Constructor & Destructor Documentation

8.198.3.1 MyFlatMutableOffset()

```
MyFlatMutableOffset::MyFlatMutableOffset ( )  [inline]
```

Creates a null Offset.

Postcondition

is_null() (p. 1876)

8.198.4 Member Function Documentation

8.198.4.1 my_primitive() [1/2]

```
int32_t MyFlatMutableOffset::my_primitive ( ) const
```

Retrieves the value for a primitive member.

Returns

The value of the member or its default value if this member doesn't exist in this Sample.

8.198.4.2 my_optional_primitive()

```
rti::flat::PrimitiveConstOffset< int32_t > MyFlatMutableOffset::my_optional_primitive ( ) const
```

Retrieves a const Offset to an optional primitive.

Unlike the non-optional **my_primitive()** (p. 1772), which accesses the integer directly, for an optional primitive it is possible to check whether it exists or not. If it doesn't exist, the Offset this function returns will be null (**is_null()** (p. 1876)).

8.198.4.3 my_primitive_array() [1/2]

```
const rti::flat::PrimitiveArrayOffset< int32_t, 10 > MyFlatMutableOffset::my_primitive_array ( )
const
```

Retrieves a const Offset to a primitive array.

8.198.4.4 my_primitive_seq() [1/2]

```
const rti::flat::PrimitiveSequenceOffset< int32_t > MyFlatMutableOffset::my_primitive_seq ( )
const
```

Retrieves a const Offset to a primitive sequence.

8.198.4.5 my_final() [1/2]

```
MyFlatFinal::ConstOffset MyFlatMutableOffset::my_final ( ) const
```

Retrieves a const Offset to a complex member.

8.198.4.6 my_final_array() [1/2]

```
rti::flat::FinalAlignedArrayOffset< MyFlatFinal::ConstOffset, 10 > MyFlatMutableOffset::my_↩
final_array ( ) const
```

Retrieves a const Offset to a complex array.

8.198.4.7 my_final_seq() [1/2]

```
rti::flat::SequenceOffset< MyFlatFinal::ConstOffset > MyFlatMutableOffset::my_final_seq ( )
const
```

Retrieves a const Offset to a complex sequence.

8.198.4.8 my_mutable() [1/2]

```
FlatMutableBar::ConstOffset MyFlatMutableOffset::my_mutable ( ) const
```

Retrieves a const Offset to a complex member.

FlatMutableBar is another arbitrary user-defined mutable FlatData type.

8.198.4.9 my_mutable_array() [1/2]

```
rti::flat::MutableArrayOffset< FlatMutableBar::ConstOffset, 10 > MyFlatMutableOffset::my_↩
mutable_array ( ) const
```

Retrieves a const Offset to a complex array.

8.198.4.10 my_mutable_seq() [1/2]

```
rti::flat::SequenceOffset< FlatMutableBar::ConstOffset > MyFlatMutableOffset::my_mutable_seq ( )
const
```

Retrieves a const Offset to a complex sequence.

8.198.4.11 my_string() [1/2]

```
const rti::flat::StringOffset MyFlatMutableOffset::my_string ( ) const
```

Retrieves a const Offset to a string.

8.198.4.12 my_string_seq() [1/2]

```
rti::flat::SequenceOffset< const  rti::flat::StringOffset > MyFlatMutableOffset::my_string_seq (
) const
```

Retrieves a const Offset to a sequence of strings.

8.198.4.13 my_primitive() [2/2]

```
bool MyFlatMutableOffset::my_primitive (
            int32_t value )
```

Sets the value of a primitive member.

8.198.4.14 my_primitive_array() [2/2]

```
rti::flat::PrimitiveArrayOffset< int32_t, 10 > MyFlatMutableOffset::my_primitive_array ( )
```

Retrieves a non-const Offset to a primitive array.

8.198.4.15 my_primitive_seq() [2/2]

```
rti::flat::PrimitiveSequenceOffset< int32_t > MyFlatMutableOffset::my_primitive_seq ( )
```

Retrieves a non-const Offset to a primitive sequence.

8.198.4.16 my_final() [2/2]

```
MyFlatFinal::Offset MyFlatMutableOffset::my_final ( )
```

Retrieves a non-const Offset to a complex member.

8.198.4.17 my_final_array() [2/2]

rti::flat::FinalAlignedArrayOffset< **MyFlatFinal::Offset**, 10 > MyFlatMutableOffset::my_final_↩
array ( )

Retrieves a non-const Offset to a complex array.

8.198.4.18 my_final_seq() [2/2]

rti::flat::SequenceOffset< **MyFlatFinal::Offset** > MyFlatMutableOffset::my_final_seq ( )

Retrieves a non-const Offset to a complex sequence.

8.198.4.19 my_mutable() [2/2]

FlatMutableBar::Offset MyFlatMutableOffset::my_mutable ( )

Retrieves a non-const Offset to a complex member.

FlatMutableBar is another arbitrary user-defined mutable FlatData type.

8.198.4.20 my_mutable_array() [2/2]

rti::flat::MutableArrayOffset< FlatMutableBar::Offset, 10 > MyFlatMutableOffset::my_mutable_↩
array ( )

Retrieves a non-const Offset to a complex array.

8.198.4.21 my_mutable_seq() [2/2]

rti::flat::SequenceOffset< FlatMutableBar::Offset > MyFlatMutableOffset::my_mutable_seq ( )

Retrieves a non-const Offset to a complex sequence.

8.198.4.22 my_string() [2/2]

```
rti::flat::StringOffset MyFlatMutableOffset::my_string ( )
```

Retrieves a non-const Offset to a string.

8.198.4.23 my_string_seq() [2/2]

```
rti::flat::SequenceOffset< rti::flat::StringOffset > MyFlatMutableOffset::my_string_seq ( )
```

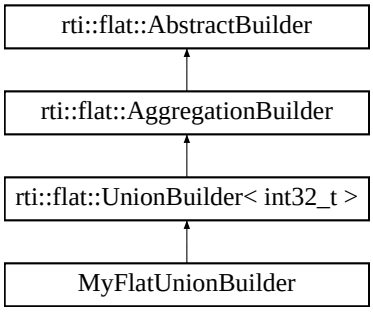
Retrieves a non-const Offset to a sequence of strings.

8.199 MyFlatUnionBuilder Class Reference

Represents the Builder for an arbitrary user-defined mutable IDL union.

`#include <Builder.hpp>`

Inheritance diagram for MyFlatUnionBuilder:



Public Types

- typedef **MyFlatUnionOffset Offset**
  The related offset type.

Public Member Functions

- **MyFlatUnionBuilder** ()
  Creates an invalid Builder.
- **Offset finish** ()
  Finishes building a member.
- **MyFlatUnion** * **finish_sample** ()
  Finishes building a sample.
- bool **add_my_primitive** (int32_t value)
  Adds a primitive member.
- **MyFlatMutableBuilder build_my_mutable** (int32_t discriminator=1)
  Builds a mutable-struct member.
- **MyFlatFinal::Offset add_my_final** ()
  Adds a final-struct member.

Additional Inherited Members

8.199.1 Detailed Description

Represents the Builder for an arbitrary user-defined mutable IDL union.

This example type represents the **Builder** (p. 409) type that **rtiddsgen** would generate for **MyFlatUnion** (p. 413) and allows creating a sample or a member of that type.

Union builders are similar to struct builders (see **MyFlatMutableBuilder** (p. 1763)), except that they only allow adding/building **one member**.

"Add" and "build" functions automatically set the discriminator value that in the IDL definition selects that member. If more than one discriminator value selects a member, the add/build function allows picking one.

8.199.2 Member Typedef Documentation

8.199.2.1 Offset

```
typedef MyFlatUnionOffset MyFlatUnionBuilder::Offset
```

The related offset type.

8.199.3 Constructor & Destructor Documentation

8.199.3.1 MyFlatUnionBuilder()

```
MyFlatUnionBuilder::MyFlatUnionBuilder ( ) [inline]
```

Creates an invalid Builder.

Postcondition

!is_valid()

Note

Top-level builders are created with **rti::flat::build_data()** (p. 411), and member builders are created with the corresponding `build_<member>` function.

8.199.4 Member Function Documentation

8.199.4.1 finish()

```
Offset MyFlatUnionBuilder::finish ( )
```

Finishes building a member.

See also

MyMutableBuilder::finish()

8.199.4.2 finish_sample()

```
MyFlatUnion * MyFlatUnionBuilder::finish_sample ( )
```

Finishes building a sample.

See also

MyMutableBuilder::finish_sample()

8.199.4.3 add_my_primitive()

```
bool MyFlatUnionBuilder::add_my_primitive (
            int32_t value )
```

Adds a primitive member.

This function automatically selects the discriminator 0, which corresponds to the member 'my_primitive' (see **MyFlat↩ Union** (p. 413))

8.199.4.4 build_my_mutable()

```
MyFlatMutableBuilder MyFlatUnionBuilder::build_my_mutable (
            int32_t discriminator = 1 )
```

Builds a mutable-struct member.

Parameters

discriminator	Allows selecting one of the possible discriminator values for this member: 1 or 2. This argument is optional: if not specified it selects the first 'case' label in the IDL definition (1).

Returns

The Builder to build this member

8.199.4.5 add_my_final()

```
MyFlatFinal::Offset MyFlatUnionBuilder::add_my_final ( )
```

Adds a final-struct member.

Returns

The Offset to the member, which can be used to set its values.

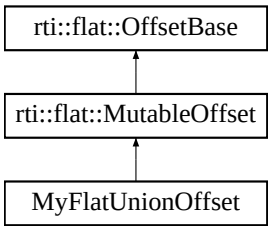
This function automatically selects the discriminator value 3, which corresponds to the member 'my_final' (see **MyFlat↩ Union** (p. 413)).

8.200 MyFlatUnionOffset Class Reference

Represents the Offset to an arbitrary user-defined FlatData mutable IDL union.

```
#include <Offset.hpp>
```

Inheritance diagram for MyFlatUnionOffset:



Public Types

- typedef MyFlatUnionConstOffset **ConstOffset**

  The equivalent read-only Offset type.

Public Member Functions

- **MyFlatUnionOffset** ()

  Creates a null Offset.
- int32_t **_d** () const

  Retrieves the union discriminator.
- int32_t **my_primitive** () const

  Retrieves the value for a primitive member.
- **MyFlatMutable::ConstOffset my_mutable** () const

  Retrieves a const Offset to a complex member.
- **MyFlatFinal::ConstOffset my_final** () const

  Retrieves a const Offset to a complex member.
- bool **my_primitive** (int32_t value)

  Sets the value for a primitive member.
- **MyFlatMutable::Offset my_mutable** ()

  Retrieves a non-const Offset to a complex member.
- **MyFlatFinal::Offset my_final** ()

  Retrieves a non-const Offset to a complex member.

8.200.1 Detailed Description

Represents the Offset to an arbitrary user-defined FlatData mutable IDL union.

This example type represents the Offset type that **rtiddsgen** would generate for **MyFlatUnion** (p. 413).

It provides accessors for each of its members, plus the discriminator **_d()** (p. 1782). Accessors can return other Offsets or primitive values.

Given a union, only one member, the one identified by **_d()** (p. 1782), can exist at a time. The discriminator cannot be modified.

8.200.2 Member Typedef Documentation

8.200.2.1 ConstOffset

```
typedef MyFlatUnionConstOffset MyFlatUnionOffset::ConstOffset
```

The equivalent read-only Offset type.

8.200.3 Constructor & Destructor Documentation

8.200.3.1 MyFlatUnionOffset()

```
MyFlatUnionOffset::MyFlatUnionOffset ( )  [inline]
```

Creates a null Offset.

Postcondition

is_null() (p. 1876)

8.200.4 Member Function Documentation

8.200.4.1 _d()

```
int32_t MyFlatUnionOffset::_d ( ) const
```

Retrieves the union discriminator.

Returns

The union discriminator, which identifies which field this union contains.

In this example:

- 0 selects **my_primitive()** (p. 1782)
- 1 and 2 select **my_mutable()** (p. 1783)
- 3 selects my_final

Any other discriminator value indicates that no member or an unknown member follows.

Note that the discriminator cannot be modified, since that would potentially change the size of this Sample by selecting a different member. The discriminator is selected during the building of a Sample (see MyUnionBuilder).

8.200.4.2 my_primitive() [1/2]

```
int32_t MyFlatUnionOffset::my_primitive ( ) const
```

Retrieves the value for a primitive member.

Returns

The value of my_primitive if this member is selected by **_d()** (p. 1782)
or its default value otherwise.

8.200.4.3 my_mutable() [1/2]

```
MyFlatMutable::ConstOffset MyFlatUnionOffset::my_mutable ( ) const
```

Retrieves a const Offset to a complex member.

Returns

The Offset to the 'my_mutable' member if selected by **_d()** (p. 1782), or a **null Offset** (p. 416) otherwise

8.200.4.4 my_final() [1/2]

```
MyFlatFinal::ConstOffset MyFlatUnionOffset::my_final ( ) const
```

Retrieves a const Offset to a complex member.

Returns

The Offset to the 'my_final' member if selected by **_d()** (p. 1782), or a **null Offset** (p. 416) otherwise

8.200.4.5 my_primitive() [2/2]

```
bool MyFlatUnionOffset::my_primitive (
            int32_t value )
```

Sets the value for a primitive member.

Precondition

_d() (p. 1782) must select 'my_primitive', otherwise this function returns false.

Returns

true if my_primitive was set or false if the operation failed.

8.200.4.6 my_mutable() [2/2]

```
MyFlatMutable::Offset MyFlatUnionOffset::my_mutable ( )
```

Retrieves a non-const Offset to a complex member.

Returns

The Offset to the 'my_mutable' member if selected by **_d()** (p. 1782), or a **null Offset** (p. 416) otherwise

8.200.4.7 my_final() [2/2]

```
MyFlatFinal::Offset MyFlatUnionOffset::my_final ( )
```

Retrieves a non-const Offset to a complex member.

Returns

The Offset to the 'my_final' member if selected by **_d()** (p. 1782), or a **null Offset** (p. 416) otherwise

8.201 NDDS_Transport_Address_t Struct Reference

Addresses are stored individually as network-ordered bytes.

`#include <transport_common_user.h>`

Public Attributes

- unsigned char **network_ordered_value** [NDDS_TRANSPORT_ADDRESS_LENGTH]

8.201.1 Detailed Description

Addresses are stored individually as network-ordered bytes.

RTI Connext addresses are numerically stored in a transport independent manner. RTI Connext uses a IPv6-compatible format, which means that the data structure to hold an **NDDS_Transport_Address_t** (p. 1784) is the same size as a data structure needed to hold an IPv6 address.

In addition, the functions provided to translate a string representation of an RTI Connext address to a value assumes that the string presentation follows the IPv6 address presentation as specified in RFC 2373.

An **NDDS_Transport_Address_t** (p. 1784) always stores the address in network-byte order (which is Big Endian).

For example, IPv4 multicast address of 225.0.0.0 is represented by

{{0,0,0,0, 0,0,0,0, 0,0,0,0, 0xE1,0,0,0}} regardless of endianness,

where 0xE1 is the 13th byte of the structure (`network_ordered_value`[12]).

8.201.2 Member Data Documentation

8.201.2.1 network_ordered_value

```
unsigned char NDDS_Transport_Address_t::network_ordered_value[NDDS_TRANSPORT_ADDRESS_LENGTH]
```

network-byte ordered (i.e., bit 0 is the most significant bit and bit 128 is the least significant bit).

8.202 NDDS_Transport_Interface_t Struct Reference

Storage for the description of a network interface used by a Transport Plugin.

```
#include <transport_common.h>
```

Public Attributes

- **NDDS_Transport_ClassId_t transport_classid**

  The transport classid of the interface.
- **NDDS_Transport_Address_t address**

  An unicast address that uniquely identifies this interface in the network specified by the transport class.
- **NDDS_Transport_Interface_Status_t status**

  The state of the interface.
- RTI_UINT16 **rank**

  Rank of the interface. Used when allow_interfaces_list Qos is set. A rank value will be assigned to each of the interfaces that match the allow_interfaces_list filter allowing an endpoint to prioritize some interfaces upon others.

8.202.1 Detailed Description

Storage for the description of a network interface used by a Transport Plugin.

8.202.2 Member Data Documentation

8.202.2.1 transport_classid

```
NDDS_Transport_ClassId_t NDDS_Transport_Interface_t::transport_classid
```

The transport classid of the interface.

8.202.2.2 address

NDDS_Transport_Address_t NDDS_Transport_Interface_t::address

An unicast address that uniquely identifies this interface in the network specified by the transport class.

8.202.2.3 status

NDDS_Transport_Interface_Status_t NDDS_Transport_Interface_t::status

The state of the interface.

8.202.2.4 rank

RTI_UINT16 NDDS_Transport_Interface_t::rank

Rank of the interface. Used when allow_interfaces_list Qos is set. A rank value will be assigned to each of the interfaces that match the allow_interfaces_list filter allowing an endpoint to prioritize some interfaces upon others.

8.203 NDDS_Transport_Property_t Struct Reference

Base configuration structure that must be inherited by derived Transport Plugin classes.

#include <transport_interface_user.h>

Public Attributes

- **NDDS_Transport_ClassId_t classid**

  The Transport-Plugin Class ID.
- RTI_INT32 **address_bit_count**

  Number of bits in a 16-byte address that are used by the transport. Should be between -128 and 128.
- RTI_INT32 **properties_bitmap**

  A bitmap that defines various properties of the transport to the RTI Connext core.
- RTI_INT32 **gather_send_buffer_count_max**

  Specifies the maximum number of buffers that RTI Connext can pass to the `send()` method of a transport plugin.
- RTI_INT32 **message_size_max**

  The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin.
- char ** **allow_interfaces_list**

  A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `allow_interfaces_list_length` > 0), allow the use of only these interfaces. If the list is empty, allow the use of all interfaces.

- RTI_INT32 **allow_interfaces_list_length**

  Number of elements in the `allow_interfaces_list`*.*
- char ** **deny_interfaces_list**

  A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `deny_interfaces_list_length` *> 0), deny the use of these interfaces.*
- RTI_INT32 **deny_interfaces_list_length**

  Number of elements in the `deny_interfaces_list`*.*
- char ** **allow_multicast_interfaces_list**

  A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `allow_multicast_interfaces_list_length` *> 0), allow the use of multicast only on these interfaces; otherwise allow the use of all the allowed interfaces.*
- RTI_INT32 **allow_multicast_interfaces_list_length**

  Number of elements in the `allow_multicast_interfaces_list`*.*
- char ** **deny_multicast_interfaces_list**

  A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `deny_multicast_interfaces_list_length` *> 0), deny the use of those interfaces for multicast.*
- RTI_INT32 **deny_multicast_interfaces_list_length**

  Number of elements in `deny_multicast_interfaces_list`*.*
- struct **NDDS_Transport_UUID transport_uuid**

  Univocally identifies a transport plugin.
- char * **thread_name_prefix**

  Prefix used to name the transport threads.
- RTI_INT32 **max_interface_count**

  Maximum number of addresses from the `allow_interfaces_list` *that will be announced.*

8.203.1 Detailed Description

Base configuration structure that must be inherited by derived Transport Plugin classes.

This structure contains properties that must be set before registration of any transport plugin with RTI Connext. The RTI Connext core will configure itself to use the plugin based on the properties set within this structure.

A transport plugin may extend from this structure to add transport-specific properties.

WARNING: The transport properties of an instance of a Transport Plugin should be considered immutable after the plugin has been created. That means the values contained in the property structure stored as a part of the transport plugin itself should not be changed. If those values are modified, the results are undefined.

8.203.2 Member Data Documentation

8.203.2.1 classid

NDDS_Transport_ClassId_t NDDS_Transport_Property_t::classid

The Transport-Plugin Class ID.

The transport plugin sets the value for this field.

Class IDs below **NDDS_TRANSPORT_CLASSID_RESERVED_RANGE** (p. 370) are reserved for RTI (Real-Time Innovations) usage.

User-defined transports should set an ID above this range.

The ID should be globally unique for each Transport-Plugin class. Transport-Plugin implementors should ensure that the class IDs do not conflict with each other amongst different Transport-Plugin classes.

Invariant

The `classid` is invariant for the lifecycle of a transport plugin.

8.203.2.2 address_bit_count

RTI_INT32 NDDS_Transport_Property_t::address_bit_count

Number of bits in a 16-byte address that are used by the transport. Should be between -128 and 128.

The transport plugin sets the value for this field.

A transport plugin should define the range of addresses that are meaningful to the plugin. It does this by setting the number of bits of an IPv6 address that will be used to designate an address in the network to which the transport plugin is connected. The sign of this field determines whether the count of bits starts on the most significant bit (negative sign) or on the least significant bit (positive sign).

For example, for an address range of 0-255, the address_bit_count should be set to 8. For the range of addresses used by IPv4 (4 bytes), it should be set to 32.

See also

Transport Class Attributes (p. 77)

8.203.2.3 properties_bitmap

```
RTI_INT32 NDDS_Transport_Property_t::properties_bitmap
```

A bitmap that defines various properties of the transport to the RTI Connext core.

Currently, the only property supported is whether or not the transport plugin will always loan a buffer when RTI Connext tries to receive a message using the plugin. This is in support of a zero-copy interface.

See also

NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED (p. 370)

8.203.2.4 gather_send_buffer_count_max

```
RTI_INT32 NDDS_Transport_Property_t::gather_send_buffer_count_max
```

Specifies the maximum number of buffers that RTI Connext can pass to the `send()` method of a transport plugin.

The transport plugin `send()` API supports a gather-send concept, where the `send()` call can take several discontiguous buffers, assemble and send them in a single message. This enables RTI Connext to send a message from parts obtained from different sources without first having to copy the parts into a single contiguous buffer.

However, most transports that support a gather-send concept have an upper limit on the number of buffers that can be gathered and sent. Setting this value will prevent RTI Connext from trying to gather too many buffers into a send call for the transport plugin.

RTI Connext requires all transport-plugin implementations to support a gather-send of least a minimum number of buffers. This minimum number is defined to be **NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_↩ COUNT_MIN** (p. 371).

If the underlying transport does not support a gather-send concept directly, then the transport plugin itself must copy the separate buffers passed into the `send()` call into a single buffer for sending or otherwise send each buffer individually. However this is done by the transport plugin, the receive_rEA() call of the destination application should assemble, if needed, all of the pieces of the message into a single buffer before the message is passed to the RTI Connext layer.

8.203.2.5 message_size_max

```
RTI_INT32 NDDS_Transport_Property_t::message_size_max
```

The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin.

If the maximum size of a message that can be sent by a transport plugin is user configurable, the transport plugin should provide a default value for this property. In any case, this value must be set before the transport plugin is registered, so that RTI Connext can properly use the plugin.

For information about the default value for different transports, see the `Core Libraries User's Manual`.

8.203.2.6 allow_interfaces_list

```
char** NDDS_Transport_Property_t::allow_interfaces_list
```

A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `allow_interfaces_list_length` > 0), allow the use of only these interfaces. If the list is empty, allow the use of all interfaces.

The "white" list restricts *reception* to a particular set of interfaces for unicast UDP.

For UDPv4 and UDPv6 transports, the resulting list also controls the interfaces over which the DomainParticipant will send multicast traffic to the remote DomainParticipants (if multicast is supported on the platform).

Note: This property does not affect the interfaces that the transport uses to send unicast data from that Domain↩ Participant. That decision is made by the OS based on the destination address.

It is up to the transport plugin to interpret the list of strings passed in.

For example, the following are acceptable strings in IPv4 format: 192.168.1.1, 192.168.1.∗, 192.168.∗, 192.∗, ether0

The left-to-right order of this list matters if you are using the max_interface_count to limit the allowable interfaces further. See `max_interface_count`.

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

The caller (user) must manage the memory of the list. The memory may be freed after the **dds::domain::Domain↩ Participant** (p. 1326) is deleted.

[default] empty list that represents all available interfaces.

See also

NDDS_Transport_Property_t::max_interface_count (p. 1793)

8.203.2.7 allow_interfaces_list_length

```
RTI_INT32 NDDS_Transport_Property_t::allow_interfaces_list_length
```

Number of elements in the `allow_interfaces_list`.

By default, `allow_interfaces_list_length` = 0, i.e. an empty list.

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

[default] 0

8.203.2.8 deny_interfaces_list

```
char** NDDS_Transport_Property_t::deny_interfaces_list
```

A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `deny_interfaces_list_length` > 0), deny the use of these interfaces.

This "black" list is applied *after* the `allow_interfaces_list` and filters out the interfaces that should not be used.

The resulting list restricts *reception* to a particular set of interfaces for unicast UDP.

For UDPv4 and UDPv6 transports, the resulting list also controls the interfaces over which the DomainParticipant will send multicast traffic to the remote DomainParticipants (if multicast is supported on the platform).

Note: This property does not affect the interfaces that the transport uses to send unicast data from that Domain↩ Participant. That decision is made by the OS based on the destination address.

It is up to the transport plugin to interpret the list of strings passed in.

For example, the following are acceptable strings in IPv4 format: 192.168.1.1, 192.168.1.∗, 192.168.∗, 192.∗, ether0

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

The caller (user) must manage the memory of the list. The memory may be freed after the **dds::domain::Domain↩ Participant** (p. 1326) is deleted.

[default] empty list that represents no denied interfaces.

8.203.2.9 deny_interfaces_list_length

```
RTI_INT32 NDDS_Transport_Property_t::deny_interfaces_list_length
```

Number of elements in the `deny_interfaces_list`.

By default, `deny_interfaces_list_length` = 0 (i.e., an empty list).

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

[default] 0

8.203.2.10 allow_multicast_interfaces_list

```
char** NDDS_Transport_Property_t::allow_multicast_interfaces_list
```

A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `allow_multicast_interfaces_list_length` > 0), allow the use of multicast only on these interfaces; otherwise allow the use of all the allowed interfaces.

This "white" list sub-selects from the allowed interfaces obtained *after* applying the `allow_interfaces_list` "white" list *and* the `deny_interfaces_list` "black" list.

After `allow_multicast_interfaces_list`, the `deny_multicast_interfaces_list` is applied. Multicast output will be sent and may be received over the interfaces in the resulting list.

If this list is empty, all the allowed interfaces will be potentially used for multicast. It is up to the transport plugin to interpret the list of strings passed in.

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

The caller (user) must manage the memory of the list. The memory may be freed after the **dds::domain::DomainParticipant** (p. 1326) is deleted.

[default] empty list that represents all available interfaces.

8.203.2.11 allow_multicast_interfaces_list_length

```
RTI_INT32 NDDS_Transport_Property_t::allow_multicast_interfaces_list_length
```

Number of elements in the `allow_multicast_interfaces_list`.

By default, `allow_multicast_interfaces_list_length` = 0 (i.e., an empty list).

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

[default] 0

8.203.2.12 deny_multicast_interfaces_list

```
char** NDDS_Transport_Property_t::deny_multicast_interfaces_list
```

A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `deny_multicast_interfaces_list_length` > 0), deny the use of those interfaces for multicast.

This "black" list is applied after `allow_multicast_interfaces_list` and filters out interfaces that should not be used for multicast.

Multicast output will be sent and may be received over the interfaces in the resulting list.

It is up to the transport plugin to interpret the list of strings passed in.

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

The caller (user) must manage the memory of the list. The memory may be freed after the **dds::domain::DomainParticipant** (p. 1326) is deleted.

[default] empty list that represents no denied interfaces.

8.203.2.13 deny_multicast_interfaces_list_length

```
RTI_INT32 NDDS_Transport_Property_t::deny_multicast_interfaces_list_length
```

Number of elements in `deny_multicast_interfaces_list`.

By default, `deny_multicast_interfaces_list_length` = 0 (i.e., an empty list).

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

[default] 0

8.203.2.14 transport_uuid

```
struct NDDS_Transport_UUID NDDS_Transport_Property_t::transport_uuid
```

Univocally identifies a transport plugin.

It represents the prefix of the participant GUID.

8.203.2.15 thread_name_prefix

```
char* NDDS_Transport_Property_t::thread_name_prefix
```

Prefix used to name the transport threads.

This field is optional.

The maximum size of this string is 8 characters.

If "thread_name_prefix" is not set by the user, RTI Connext creates the following prefix: 'r' + 'Tr' + participant identifier + '\0'.

Where 'r' indicates this is a thread from RTI, 'Tr' indicates the thread is related to a transport, and participant identifier contains 5 characters as follows:

If participant_name is set: The participant identifier will be the first 3 characters and the last 2 characters of the participant_name.

If participant_name is not set, then the identifier is computed as domain_id (3 characters) followed by participant_id (2 characters).

If participant_name is not set and the participant_id is set to -1 (defalt value), then the participant identifier is computed as the last 5 digits of the rtps_instance_id in the participant GUID.

8.203.2.16 max_interface_count

```
RTI_INT32 NDDS_Transport_Property_t::max_interface_count
```

Maximum number of addresses from the `allow_interfaces_list` that will be announced.

By default, `max_interface_count` = LENGTH_UNLIMITED: all the interfaces are announced.

This feature is useful if you want to control the network interfaces on which your DomainParticipants receive data. For example, if you have one wired and one wireless interface both up and running, and `max_interface_count` is set to 1, the DomainParticipant will receive data over the interface you list first in the `allow_interfaces_list` -for example, the wired one.

RTI Connext selects the preferred interface(s) by iterating over the list of allowed interfaces until the first `max_↩ interfaces_count` of active interfaces encountered are announced. The order of iteration is left to right as specified in the `allow_interfaces_list` setting.

This setting applies only if the `allow_interfaces_list` is not empty.

The `max_interface_count` setting does not consider end-to-end connectivity to select interfaces. The decision is based purely on whether interfaces are up or down in a node. Therefore, this feature is not intended to be used in the following scenarios:

- A DomainParticipant is not reachable by other DomainParticipants in all the interfaces in the `allow_↩ interfaces_list`. This could occur if the DomainParticipant is in different subnets, and some of these subnets cannot be reached by other DomainParticipants.
- End-to-end connectivity issues lead to situations in which the interfaces selected after applying `max_↩ interface_count` cannot be reached by other DomainParticipants.

For UDPv4 and UDPv6 transports, this feature also affects multicast traffic by limiting the interfaces over which a DomainParticipant sends multicast traffic. The (allow/deny)_multicast_interfaces_list applies to the interfaces selected by using the `max_interfaces_count` property.

Note: If a pattern string in the `allow_interfaces_list` matches multiple interface addresses, and `max_↩ interface_count` is set to a finite value, the order for the matching allowed interfaces is decided based on the order in which the operating system provides these interfaces.

[default] LENGTH_UNLIMITED

See also

NDDS_Transport_Property_t::allow_interfaces_list (p. 1789)

8.204 NDDS_Transport_Shmem_Property_t Struct Reference

Subclass of **NDDS_Transport_Property_t** (p. 1786) allowing specification of parameters that are specific to the shared-memory transport.

```
#include <transport_shmem.h>
```

Public Attributes

- struct **NDDS_Transport_Property_t parent**

  Generic properties of all transport plugins.
- RTI_INT32 **received_message_count_max**

  Number of messages that can be buffered in the receive queue.
- RTI_INT32 **receive_buffer_size**

  The total number of bytes that can be buffered in the receive queue.
- RTIBool **enable_udp_debugging**

  Enables UDP debugging when using shared memory.
- **NDDS_Transport_Address_t udp_debugging_address**

  IP address to which shared memory traffic will be published if ShmemTransport_Property_t::enable_udp_debugging is set to '1'.
- **NDDS_Transport_Port_t udp_debugging_port**

  Port to which shared memory traffic will be published if ShmemTransport_Property_t::enable_udp_debugging is set to '1'.

8.204.1 Detailed Description

Subclass of **NDDS_Transport_Property_t** (p. 1786) allowing specification of parameters that are specific to the shared-memory transport.

See also

Transport_Support::set_builtin_transport_property()

8.204.2 Member Data Documentation

8.204.2.1 parent

`struct` **`NDDS_Transport_Property_t`** `NDDS_Transport_Shmem_Property_t::parent`

Generic properties of all transport plugins.

8.204.2.2 received_message_count_max

`RTI_INT32 NDDS_Transport_Shmem_Property_t::received_message_count_max`

Number of messages that can be buffered in the receive queue.

This does not guarantee that the Transport-Plugin will actually be able to buffer `received_message_count_max` messages of the maximum size set in **NDDS_Transport_Property_t::message_size_max** (p. 1789). The total number of bytes that can be buffered for a transport plug-in is actually controlled by `receive_buffer_size`.

See also

NDDS_Transport_Property_t (p. 1786), **NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_↩ MAX_DEFAULT** (p. 384)

8.204.2.3 receive_buffer_size

```
RTI_INT32 NDDS_Transport_Shmem_Property_t::receive_buffer_size
```

The total number of bytes that can be buffered in the receive queue.

This number controls how much memory is allocated by the plugin for the receive queue. The actual number of bytes allocated is:

```
size = receive_buffer_size + message_size_max +
       received_message_count_max * fixedOverhead
```

where `fixedOverhead` is some small number of bytes used by the queue data structure. The following rules are noted:

- `receive_buffer_size` < `message_size_max` * `received_message_count_max`, then the transport plugin will not be able to store `received_message_count_max` messages of size `message_size_max`.

- `receive_buffer_size` > `message_size_max` * `received_message_count_max`, then there will be memory allocated that cannot be used by the plugin and thus wasted.

To optimize memory usage, the user is allowed to specify a size for the receive queue to be less than that required to hold the maximum number of messages which are all of the maximum size.

In most situations, the average message size may be far less than the maximum message size. So for example, if the maximum message size is 64 K bytes, and the user configures the plugin to buffer at least 10 messages, then 640 K bytes of memory would be needed if all messages were 64 K bytes. Should this be desired, then `receive_buffer_size` should be set to 640 K bytes.

However, if the average message size is only 10 K bytes, then the user could set the `receive_buffer_size` to 100 K bytes. This allows the user to optimize the memory usage of the plugin for the average case and yet allow the plugin to handle the extreme case.

NOTE, the queue will always be able to hold 1 message of `message_size_max` bytes, no matter what the value of receive_buffer_size is.

See also

NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT (p. 385)

8.204.2.4 enable_udp_debugging

```
RTIBool NDDS_Transport_Shmem_Property_t::enable_udp_debugging
```

Enables UDP debugging when using shared memory.

If set to '1', all shared memory traffic will be published to ShmemTransport_Property_t::udp_debugging_address :: ShmemTransport_Property_t::udp_debugging_port.

[default] 0

8.204.2.5 udp_debugging_address

NDDS_Transport_Address_t NDDS_Transport_Shmem_Property_t::udp_debugging_address

IP address to which shared memory traffic will be published if ShmemTransport_Property_t::enable_udp_debugging is set to '1'.

[default] 239.255.1.2

8.204.2.6 udp_debugging_port

NDDS_Transport_Port_t NDDS_Transport_Shmem_Property_t::udp_debugging_port

Port to which shared memory traffic will be published if ShmemTransport_Property_t::enable_udp_debugging is set to '1'.

[default] 7399

8.205 NDDS_Transport_UDP_WAN_CommPortsMappingInfo Struct Reference

Type for storing UDP WAN communication ports.

`#include <transport_udp.h>`

Public Attributes

- **NDDS_Transport_Port_t rtps_port**
  RTPS port.
- **NDDS_Transport_UDP_Port host_port**
  Host port.
- **NDDS_Transport_UDP_Port public_port**
  Public port.

8.205.1 Detailed Description

Type for storing UDP WAN communication ports.

8.205.2 Member Data Documentation

8.205.2.1 rtps_port

NDDS_Transport_Port_t NDDS_Transport_UDP_WAN_CommPortsMappingInfo::rtps_port

RTPS port.

8.205.2.2 host_port

NDDS_Transport_UDP_Port NDDS_Transport_UDP_WAN_CommPortsMappingInfo::host_port

Host port.

8.205.2.3 public_port

NDDS_Transport_UDP_Port NDDS_Transport_UDP_WAN_CommPortsMappingInfo::public_port

Public port.

8.206 NDDS_Transport_UDPv4_Property_t Struct Reference

Configurable IPv4/UDP Transport-Plugin properties.

#include <transport_udpv4.h>

Public Attributes

- struct **NDDS_Transport_Property_t parent**
  Generic properties of all transport plugins.
- RTI_INT32 **send_socket_buffer_size**
  Size in bytes of the send buffer of a socket used for sending.
- RTI_INT32 **recv_socket_buffer_size**
  Size in bytes of the receive buffer of a socket used for receiving.
- RTI_INT32 **unicast_enabled**
  Allows the transport plugin to use unicast for sending and receiving.
- RTI_INT32 **multicast_enabled**
  Allows the transport plugin to use multicast for sending and receiving.
- RTI_INT32 **multicast_ttl**
  Value for the time-to-live parameter for all multicast sends using this plugin.
- RTI_INT32 **multicast_loopback_disabled**
  Prevents the transport plugin from putting multicast packets onto the loopback interface.

- RTI_INT32 **ignore_loopback_interface**

  Prevents the transport plugin from using the IP loopback interface.
- RTI_INT32 **ignore_nonup_interfaces**

  [DEPRECATED] *Prevents the transport plugin from using a network interface that is not reported as UP by the operating system.*
- RTI_INT32 **ignore_nonrunning_interfaces**

  Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.
- RTI_INT32 **no_zero_copy**

  [DEPRECATED] *Prevents the transport plugin from doing a zero copy.*
- RTI_INT32 **send_blocking**

  Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.
- RTI_INT32 **use_checksum**

  Configures whether to send UDP checksum.
- RTI_UINT32 **transport_priority_mask**

  Set mask for use of transport priority field.
- RTI_INT32 **transport_priority_mapping_low**

  Set low value of output range to IPv4 TOS.
- RTI_INT32 **transport_priority_mapping_high**

  Set high value of output range to IPv4 TOS.
- RTI_INT32 **send_ping**

  Configures whether to send PING messages.
- RTI_INT32 **force_interface_poll_detection**

  Forces the interface tracker to use a polling mechanism to detect changes on the UDPv4 interfaces.
- RTI_UINT32 **interface_poll_period**

  Specifies the period in milliseconds to query for changes in the state of all the interfaces.
- RTI_INT32 **reuse_multicast_receive_resource**

  Controls whether or not to reuse multicast receive resources.
- RTI_INT32 **protocol_overhead_max**

  Maximum size in bytes of protocol overhead, including headers.
- RTI_INT32 **disable_interface_tracking**

  Disables detection of network interface changes.
- RTI_UINT32 **join_multicast_group_timeout**

  [Windows only] Defines how much time (milliseconds) to wait to join a multicast group address when a new interface is detected.
- char * **public_address**

  Public IP address associated with the transport instantiation.
- RTI_INT32 **port_offset**

  Port offset to allow multiple instances of UDPv4 transports in the same ***dds::domain::DomainParticipant*** *(*p. *1326)**.*

8.206.1 Detailed Description

Configurable IPv4/UDP Transport-Plugin properties.

You can modify the properties in this structure to configure the plugin. However, you must set the properties before the plugin is instantiated.

See also

Transport_Support::set_builtin_transport_property()

8.206.2 Member Data Documentation

8.206.2.1 parent

`struct` **NDDS_Transport_Property_t** NDDS_Transport_UDPv4_Property_t::parent

Generic properties of all transport plugins.

8.206.2.2 send_socket_buffer_size

RTI_INT32 NDDS_Transport_UDPv4_Property_t::send_socket_buffer_size

Size in bytes of the send buffer of a socket used for sending.

On most operating systems, `setsockopt()` will be called to set the SENDBUF to the value of this parameter.

This value must be greater than or equal to **NDDS_Transport_Property_t::message_size_max** (p. 1789). Its maximum value is dependent on the operating system.

By default, it will be set to UDPv4Transport_SEND_SOCKET_BUFFER_SIZE_DEFAULT.

The maximum value for this property depends on the operating system.

Valid values are:

- UDPv4Transport_SEND_SOCKET_BUFFER_SIZE_DEFAULT: `setsockopt()` (or equivalent) will be called to size the send buffer of the socket with the default value.
- UDPv4Transport_SOCKET_BUFFER_SIZE_OS_DEFAULT: `setsockopt()` (or equivalent) will not be called to size the send buffer of the socket. RTI Connext will use the OS default.
- UDPv4Transport_SOCKET_BUFFER_SIZE_OS_MAX: RTI Connext will retrieve the maximum socket buffer size allowed by the operating system and use that value for the send buffer size of the socket.
- Positive integer: `setsockopt()` (or equivalent) will be called to size the send buffer of the socket with the provided value. If this value exceeds the maximum allowed by the operating system, the operating system's maximum value will be used instead.

8.206.2.3 recv_socket_buffer_size

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::recv_socket_buffer_size
```

Size in bytes of the receive buffer of a socket used for receiving.

On most operating systems, `setsockopt()` will be called to set the RECVBUF to the value of this parameter.

This value must be greater than or equal to **NDDS_Transport_Property_t::message_size_max** (p. 1789). Its maximum value is dependent on the operating system.

By default, it will be set to UDPv4Transport_RECV_SOCKET_BUFFER_SIZE_DEFAULT.

The maximum value for this property depends on the operating system.

Valid values are:

- UDPv4Transport_RECV_SOCKET_BUFFER_SIZE_DEFAULT: `setsockopt()` (or equivalent) will be called to size the receive buffer of the socket with the default value.
- UDPv4Transport_SOCKET_BUFFER_SIZE_OS_DEFAULT: `setsockopt()` (or equivalent) will not be called to size the receive buffer of the socket. RTI Connext will use the OS default.
- UDPv4Transport_SOCKET_BUFFER_SIZE_OS_MAX: RTI Connext will retrieve the maximum socket buffer size allowed by the operating system and use that value for the receive buffer size of the socket.
- Positive integer: `setsockopt()` (or equivalent) will be called to size the receive buffer of the socket with the provided value. If this value exceeds the maximum allowed by the operating system, the operating system's maximum value will be used instead.

8.206.2.4 unicast_enabled

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::unicast_enabled
```

Allows the transport plugin to use unicast for sending and receiving.

This value turns unicast UDP on (if set to 1) or off (if set to 0) for this plugin. By default, it will be turned on (1). Also by default, the plugin will use all the allowed network interfaces that it finds up and running when the plugin is instantiated.

8.206.2.5 multicast_enabled

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::multicast_enabled
```

Allows the transport plugin to use multicast for sending and receiving.

This value turns multicast UDP on (if set to 1) or off (if set to 0) for this plugin. By default, it will be turned on (1) for those platforms that support multicast. Also by default, the plugin will use all the network interfaces allowed for multicast that it finds up and running when the plugin is instantiated.

8.206.2.6 multicast_ttl

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::multicast_ttl
```

Value for the time-to-live parameter for all multicast sends using this plugin.

This value is used to set the TTL of multicast packets sent by this transport plugin.

[default] 1

8.206.2.7 multicast_loopback_disabled

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::multicast_loopback_disabled
```

Prevents the transport plugin from putting multicast packets onto the loopback interface.

If multicast loopback is disabled (this value is set to 1), then when sending multicast packets, RTI Connext will *not* put a copy of the packets on the loopback interface. This prevents applications on the same node (including itself) from receiving those packets.

This value is set to 0 by default, meaning multicast loopback is *enabled*.

Disabling multicast loopback (setting this value to 1) may result in minor performance gains when using multicast.

[NOTE: Windows CE systems do not support multicast loopback. This field is ignored for Windows CE targets.]

8.206.2.8 ignore_loopback_interface

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::ignore_loopback_interface
```

Prevents the transport plugin from using the IP loopback interface.

Currently three values are allowed:

- **0**: Forces local traffic to be sent over loopback, even if a more efficient transport (such as shared memory) is installed (in which case traffic will be sent over both transports).
- **1**: Disables local traffic via this plugin. The IP loopback interface is not used, even if no NICs are discovered. This is useful when you want applications running on the same node to use a more efficient plugin (such as shared memory) instead of the IP loopback.
- **-1**: Automatic. Lets RTI Connext decide between the above two choices.

The current "automatic" (-1) RTI Connext policy is as follows:

- If a shared memory transport plugin is available for local traffic and there is a locator on the initial peers list that can use shared memory, the effective value is 1 (i.e., disable UPV4 local traffic).
- Otherwise, the effective value is 0 (i.e., use UDPv4 for local traffic also).

[default] -1 Automatic RTI Connext policy based on availability of the shared memory transport.

8.206.2.9 ignore_nonup_interfaces

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::ignore_nonup_interfaces
```

[DEPRECATED] Prevents the transport plugin from using a network interface that is not reported as UP by the operating system.

DEPRECATED: this property has no effect. Non-UP interfaces are ignored until they change their status to UP, unless **NDDS_Transport_UDPv4_Property_t::disable_interface_tracking** (p. 1807) is set to 1, in which case interfaces are ignored even if they change their status at a later point.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows you to configure the transport to start using even the interfaces that were not reported as UP.

Two values are allowed:

- **0**: Allow the use of interfaces that were not reported as UP.
- **1**: Do not use interfaces that were not reported as UP.

[default] 1

8.206.2.10 ignore_nonrunning_interfaces

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::ignore_nonrunning_interfaces
```

Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows the same check to be extended to the IFF_↩ RUNNING flag implemented by some operating systems. The RUNNING flag is defined to mean that "all resources are allocated," and may be off if there is no link detected, e.g., the network cable is unplugged.

Two values are allowed:

- **0**: Do not check the RUNNING flag when enumerating interfaces, just make sure the interface is UP.
- **1**: Check the flag when enumerating interfaces, and ignore those that are not reported as RUNNING. This can be used on some operating systems to cause the transport to ignore interfaces that are enabled but not connected to the network.

[default] 1 (i.e., check RUNNING flag)

8.206.2.11 no_zero_copy

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::no_zero_copy
```

[DEPRECATED] Prevents the transport plugin from doing a zero copy.

DEPRECATED: This property has no effect. By default, this plugin will use the zero copy on OSes that offer it. While this is good for performance, it may sometimes tax the OS resources in a manner that cannot be overcome by the application.

The best example is if the hardware/device driver lends the buffer to the application itself. If the application does not return the loaned buffers soon enough, the node may err or malfunction. In case you cannot reconfigure the H/W, device driver, or the OS to allow the zero copy feature to work for your application, you may have no choice but to turn off zero copy use.

By default this is set to 0, so RTI Connext will use the zero-copy API if offered by the OS.

8.206.2.12 send_blocking

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::send_blocking
```

Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

Currently two values are defined:

- **NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS:** Sockets are blocking (default socket options for the operating system).
- **NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER:** Sockets are modified to make them non-blocking. THIS MAY CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

[default] NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS

8.206.2.13 use_checksum

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::use_checksum
```

Configures whether to send UDP checksum.

This property specifies whether the UDP checksum will be computed. On Windows and Linux, the UDP protocol will not set the checksum when use_checksum is set to 0. This is useful when RTPS protocol statistics related to corrupted messages need to be collected through the API **dds::domain::DomainParticipant::participant_protocol_status** (p. 1358).

[default] 1 (enabled)

8.206.2.14 transport_priority_mask

```
RTI_UINT32 NDDS_Transport_UDPv4_Property_t::transport_priority_mask
```

Set mask for use of transport priority field.

This is used in conjunction with **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_low** (p. 1805) and **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_high** (p. 1805) to define the mapping from the DDS transport priority (see **TRANSPORT_PRIORITY** (p. 550)) to the IPv4 TOS field. Defines a contiguous region of bits in the 32-bit transport priority value that is used to generate values for the IPv4 TOS field on an outgoing socket.

For example, the value 0x0000ff00 causes bits 9-16 (8 bits) to be used in the mapping. The value will be scaled from the mask range (0x0000 - 0xff00 in this case) to the range specified by low and high.

If the mask is set to zero, then the transport will not set IPv4 TOS for send sockets.

[default] 0.

8.206.2.15 transport_priority_mapping_low

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_low
```

Set low value of output range to IPv4 TOS.

This is used in conjunction with **NDDS_Transport_UDPv4_Property_t::transport_priority_mask** (p. 1804) and **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_high** (p. 1805) to define the mapping from the DDS transport priority to the IPv4 TOS field. Defines the low value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0.

8.206.2.16 transport_priority_mapping_high

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_high
```

Set high value of output range to IPv4 TOS.

This is used in conjunction with **NDDS_Transport_UDPv4_Property_t::transport_priority_mask** (p. 1804) and **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_low** (p. 1805) to define the mapping from the DDS transport priority to the IPv4 TOS field. Defines the high value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0xff.

8.206.2.17 send_ping

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::send_ping
```

Configures whether to send PING messages.

This property specifies whether to send a PING message before commencing the discovery process. On certain operating systems or with certain switches the initial UDP packet, while configuring the ARP table, can unfortunately be dropped. To avoid dropping the initial RTPS discovery sample, a PING message is sent to preconfigure the ARP table in those environments.

[default] 1 (enabled)

8.206.2.18 force_interface_poll_detection

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::force_interface_poll_detection
```

Forces the interface tracker to use a polling mechanism to detect changes on the UDPv4 interfaces.

When possible, the detection of an IP address change is done asynchronously using the APIs offered by the underlying OS. By setting this property on those OSes, the use of a polling mechanism to detect changes can be forced.

[default] 0 (disabled).

8.206.2.19 interface_poll_period

```
RTI_UINT32 NDDS_Transport_UDPv4_Property_t::interface_poll_period
```

Specifies the period in milliseconds to query for changes in the state of all the interfaces.

When possible, the detection of an IP address change is done asynchronously using the APIs offered by the underlying OS. If there is no mechanism to do that, the detection will use a polling strategy where the polling period can be configured by setting this property.

[default] 500 milliseconds.

8.206.2.20 reuse_multicast_receive_resource

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::reuse_multicast_receive_resource
```

Controls whether or not to reuse multicast receive resources.

Setting this to 0 (FALSE) prevents multicast crosstalk by uniquely configuring a port and creating a receive thread for each multicast group address.

[default] 1.

8.206.2.21 protocol_overhead_max

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::protocol_overhead_max
```

Maximum size in bytes of protocol overhead, including headers.

This value is the maximum size, in bytes, of protocol-related overhead. Normally, the overhead accounts for UDP and IP headers. The default value is set to accommodate the most common UDP/IP header size.

Note that when **NDDS_Transport_Property_t::message_size_max** (p. 1789) plus this overhead is larger than the UDPv4 maximum message size (65535 bytes), the middleware will automatically reduce the effective message_size↩ _max, to 65535 minus this overhead.

[default] 28.

See also

NDDS_Transport_Property_t::message_size_max (p. 1789)

8.206.2.22 disable_interface_tracking

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::disable_interface_tracking
```

Disables detection of network interface changes.

By default, network interface changes are propagated in the form of locators to other applications. This is done to support IP mobility scenarios.

For example, you could start a RTI Connext application with Wi-Fi and move to a wired connection. In order to continue communicating with other applications, this interface change has to be propagated.

In RTI Connext 5.2 (the initial release) and earlier versions of the product, IP mobility scenarios were not supported. 5.2 applications will report errors if they detect locator changes in a DataWriter or DataReader.

You can disable the notification and propagation of interface changes by setting this property to 1.

This way, an interface change in a newer application will not trigger errors in an application running 5.2 or earlier. Of course, this will prevent the new application from being able to detect network interface changes.

[default] 0

8.206.2.23 join_multicast_group_timeout

```
RTI_UINT32 NDDS_Transport_UDPv4_Property_t::join_multicast_group_timeout
```

[Windows only] Defines how much time (milliseconds) to wait to join a multicast group address when a new interface is detected.

On Windows, a network interface may be detected before it is allowed to join a multicast group address. This property adjusts how much time (milliseconds) to wait for the ADD_MEMBERSHIP multicast operation to succeed before withdrawing.

[default] 5000

8.206.2.24 public_address

```
char* NDDS_Transport_UDPv4_Property_t::public_address
```

Public IP address associated with the transport instantiation.

Setting the public IP address is only necessary to support communication over WAN that involves Network Address Translation (NAT).

Typically, the address is the public address of the IP NAT router that provides access to the WAN.

By default, the **dds::domain::DomainParticipant** (p. 1326) creating the transport will announce the IP addresses obtained from the NICs to other DomainParticipants in the system.

When this property is set, the DomainParticipant will announce the IP address corresponding to the property value instead of the LAN IP addresses associated with the NICs.

Note 1: Setting this property is necessary, but is not a sufficient condition for sending and receiving data over the WAN. You must also configure the IP NAT router to allow UDP traffic and to map the public IP address specified by this property to the DomainParticipant's private LAN IP address. This is typically done with one of the following mechanisms:

- Port Forwarding: You must map the private ports used to receive discovery and user data traffic to the corresponding public ports (see **rti::core::RtpsWellKnownPorts** (p. 2258)). Public and private ports must be the same since the transport does not allow you to change the mapping.
- 1:1 NAT: You must add a 1:1 NAT entry that maps the public IP address specified in this property to the private LAN IP address of the DomainParticipant.

Note 2: By setting this property, the **dds::domain::DomainParticipant** (p. 1326) only announces its public IP address to other DomainParticipants. Therefore, communication with DomainParticipants within the LAN that are running on different nodes will not work unless the NAT router is configured to enable NAT reflection (hairpin NAT).

There is another way to achieve simultaneous communication with DomainParticipants running in the LAN and WAN, that does not require hairpin NAT. This way uses a gateway application such as RTI Routing Service to provide access to the WAN.

[default] null (the transport uses the IP addresses obtained from the NICs)

8.206.2.25 port_offset

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::port_offset
```

Port offset to allow multiple instances of UDPv4 transports in the same **dds::domain::DomainParticipant** (p. 1326).

By default, it is not possible to have multiple instances of the UDPv4 transport in the same **dds::domain::Domain↩ Participant** (p. 1326). This is because all instances will try to bind to the same UDP port(s) for receiving data.

With this property, you can specify an offset that will be added to the RTPS port(s) to determine the UDP port(s) to bind to. This way, you can have multiple instances of the UDPv4 transport in the same **dds::domain::DomainParticipant** (p. 1326).

An RTPS port will be mapped to an UDP port as follows: UDP port = RTPS port + port_offset.

[default] 0

8.207 NDDS_Transport_UDPv4_WAN_Property_t Struct Reference

Configurable IPv4/UDP WAN Transport-Plugin properties.

`#include <transport_udpv4.h>`

Public Attributes

- struct **NDDS_Transport_Property_t parent**

  Generic properties of all transport plugins.
- RTI_INT32 **send_socket_buffer_size**

  Size in bytes of the send buffer of a socket used for sending.
- RTI_INT32 **recv_socket_buffer_size**

  Size in bytes of the receive buffer of a socket used for receiving.
- RTI_INT32 **ignore_loopback_interface**

  Prevents the transport plugin from using the IP loopback interface.
- RTI_INT32 **ignore_nonup_interfaces**

  ***[DEPRECATED]** Prevents the transport plugin from using a network interface that is not reported as UP by the operating system.*
- RTI_INT32 **ignore_nonrunning_interfaces**

  Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.
- RTI_INT32 **no_zero_copy**

  ***[DEPRECATED]** Prevents the transport plugin from doing a zero copy.*
- RTI_INT32 **send_blocking**

  Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.
- RTI_INT32 **use_checksum**

  Configures whether to send UDP checksum.
- RTI_UINT32 **transport_priority_mask**

  Set mask for use of transport priority field.
- RTI_INT32 **transport_priority_mapping_low**

  Set low value of output range to IPv4 TOS.
- RTI_INT32 **transport_priority_mapping_high**

  Set high value of output range to IPv4 TOS.
- RTI_INT32 **send_ping**

  Configures whether to send PING messages.
- RTI_INT32 **force_interface_poll_detection**

  Forces the interface tracker to use a polling mechanism to detect changes on the UDPv4 interfaces.
- RTI_UINT32 **interface_poll_period**

  Specifies the period in milliseconds to query for changes in the state of all the interfaces.
- RTI_INT32 **protocol_overhead_max**

  Maximum size in bytes of protocol overhead, including headers.
- RTI_INT32 **disable_interface_tracking**

  Disables detection of network interface changes.
- char * **public_address**

  Public IP address associated with the transport instantiation.
- struct **NDDS_Transport_UDP_WAN_CommPortsMappingInfo** * **comm_ports_list**

Configures the public and private UDP ports that a transport instance uses to receive/send RTPS data.

- RTI_INT32 **comm_ports_list_length**

  Number of elements in the ***NDDS_Transport_UDPv4_WAN_Property_t::comm_ports_list*** *(p. 1816).*

- RTI_INT32 **port_offset**

  Port offset to allow coexistence with built-in UDPv4 transport.

- RTI_UINT32 **binding_ping_period**

  Specifies the period in milliseconds at which BINDING PINGS messages are sent to keep NAT mappings open.

8.207.1 Detailed Description

Configurable IPv4/UDP WAN Transport-Plugin properties.

You can modify the properties in this structure to configure the plugin. However, you must set the properties before the plugin is instantiated.

See also

Transport_Support::set_builtin_transport_property()

8.207.2 Member Data Documentation

8.207.2.1 parent

`struct` **NDDS_Transport_Property_t** `NDDS_Transport_UDPv4_WAN_Property_t::parent`

Generic properties of all transport plugins.

8.207.2.2 send_socket_buffer_size

`RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::send_socket_buffer_size`

Size in bytes of the send buffer of a socket used for sending.

See also

NDDS_Transport_UDPv4_Property_t::send_socket_buffer_size (p. 1800)

8.207.2.3 recv_socket_buffer_size

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::recv_socket_buffer_size
```

Size in bytes of the receive buffer of a socket used for receiving.

See also

NDDS_Transport_UDPv4_Property_t::recv_socket_buffer_size (p. 1800)

8.207.2.4 ignore_loopback_interface

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::ignore_loopback_interface
```

Prevents the transport plugin from using the IP loopback interface.

Currently three values are allowed:

- **0**: Forces local traffic to be sent over loopback, even if a more efficient transport (such as shared memory) is installed (in which case traffic will be sent over both transports).
- **1**: Disables local traffic via this plugin. The IP loopback interface is not used, even if no NICs are discovered. This is useful when you want applications running on the same node to use a more efficient plugin (such as shared memory) instead of the IP loopback.
- **-1**: Automatic. Lets RTI Connext decide between the above two choices.

The current "automatic" (-1) RTI Connext policy is as follows:

- If a shared memory transport plugin is available for local traffic and there is a locator on the initial peers list that can use shared memory, the effective value is 1 (i.e., disable UPV4 local traffic).
- Otherwise, the effective value is 0 (i.e., use UDPv4 for local traffic also).

[default] -1 Automatic RTI Connext policy based on availability of the shared memory transport.

8.207.2.5 ignore_nonup_interfaces

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::ignore_nonup_interfaces
```

[DEPRECATED] Prevents the transport plugin from using a network interface that is not reported as UP by the operating system.

DEPRECATED: this property has no effect. Non-UP interfaces are ignored until they change their status to UP, unless **NDDS_Transport_UDPv4_WAN_Property_t::disable_interface_tracking** (p. 1815) is set to 1, in which case interfaces are ignored even if they change their status at a later point.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows you to configure the transport to start using even the interfaces that were not reported as UP.

Two values are allowed:

- **0**: Allow the use of interfaces that were not reported as UP.
- **1**: Do not use interfaces that were not reported as UP.

[default] 1

8.207.2.6 ignore_nonrunning_interfaces

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::ignore_nonrunning_interfaces
```

Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows the same check to be extended to the IFF_↩ RUNNING flag implemented by some operating systems. The RUNNING flag is defined to mean that "all resources are allocated," and may be off if there is no link detected, e.g., the network cable is unplugged.

Two values are allowed:

- **0**: Do not check the RUNNING flag when enumerating interfaces, just make sure the interface is UP.
- **1**: Check the flag when enumerating interfaces, and ignore those that are not reported as RUNNING. This can be used on some operating systems to cause the transport to ignore interfaces that are enabled but not connected to the network.

[default] 1 (i.e., check RUNNING flag)

8.207.2.7 no_zero_copy

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::no_zero_copy
```

[DEPRECATED] Prevents the transport plugin from doing a zero copy.

DEPRECATED: This property has no effect. By default, this plugin will use the zero copy on OSes that offer it. While this is good for performance, it may sometimes tax the OS resources in a manner that cannot be overcome by the application.

The best example is if the hardware/device driver lends the buffer to the application itself. If the application does not return the loaned buffers soon enough, the node may err or malfunction. In case you cannot reconfigure the H/W, device driver, or the OS to allow the zero copy feature to work for your application, you may have no choice but to turn off zero copy use.

By default this is set to 0, so RTI Connext will use the zero-copy API if offered by the OS.

8.207.2.8 send_blocking

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::send_blocking
```

Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

Currently two values are defined:

- **NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS:** Sockets are blocking (default socket options for the operating system).
- **NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER:** Sockets are modified to make them non-blocking. THIS MAY CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

[default] NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS

This property is not supported on Windows when using the Real-Time WAN Transport. The send sockets cannot be configured to be non-blocking because Windows does not provide a way to configure a socket to only be non-blocking for send operations but not receive operations.

8.207.2.9 use_checksum

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::use_checksum
```

Configures whether to send UDP checksum.

This property specifies whether the UDP checksum will be computed. On Windows and Linux, the UDP protocol will not set the checksum when use_checksum is set to 0. This is useful when RTPS protocol statistics related to corrupted messages need to be collected through the API **dds::domain::DomainParticipant::participant_protocol↩ _status** (p. 1358).

[default] 1 (enabled)

8.207.2.10 transport_priority_mask

```
RTI_UINT32 NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mask
```

Set mask for use of transport priority field.

This is used in conjunction with **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_low** (p. 1814) and **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_high** (p. 1814) to define the mapping from the DDS transport priority (see **TRANSPORT_PRIORITY** (p. 550)) to the IPv4 TOS field. Defines a contiguous region of bits in the 32-bit transport priority value that is used to generate values for the IPv4 TOS field on an outgoing socket.

For example, the value 0x0000ff00 causes bits 9-16 (8 bits) to be used in the mapping. The value will be scaled from the mask range (0x0000 - 0xff00 in this case) to the range specified by low and high.

If the mask is set to zero, then the transport will not set IPv4 TOS for send sockets.

[default] 0.

8.207.2.11 transport_priority_mapping_low

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_low
```

Set low value of output range to IPv4 TOS.

This is used in conjunction with **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mask** (p. 1813) and **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_high** (p. 1814) to define the mapping from the DDS transport priority to the IPv4 TOS field. Defines the low value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0.

8.207.2.12 transport_priority_mapping_high

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_high
```

Set high value of output range to IPv4 TOS.

This is used in conjunction with **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mask** (p. 1813) and **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_low** (p. 1814) to define the mapping from the DDS transport priority to the IPv4 TOS field. Defines the high value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0xff.

8.207.2.13 send_ping

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::send_ping
```

Configures whether to send PING messages.

This property specifies whether to send a PING message before commencing the discovery process. On certain operating systems or with certain switches the initial UDP packet, while configuring the ARP table, can unfortunately be dropped. To avoid dropping the initial RTPS discovery sample, a PING message is sent to preconfigure the ARP table in those environments.

[default] 1 (enabled)

8.207.2.14 force_interface_poll_detection

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::force_interface_poll_detection
```

Forces the interface tracker to use a polling mechanism to detect changes on the UDPv4 interfaces.

When possible, the detection of an IP address change is done asynchronously using the APIs offered by the underlying OS. By setting this property on those OSes, the use of a polling mechanism to detect changes can be forced.

[default] 0 (disabled).

8.207.2.15 interface_poll_period

```
RTI_UINT32 NDDS_Transport_UDPv4_WAN_Property_t::interface_poll_period
```

Specifies the period in milliseconds to query for changes in the state of all the interfaces.

When possible, the detection of an IP address change is done asynchronously using the APIs offered by the underlying OS. If there is no mechanism to do that, the detection will use a polling strategy where the polling period can be configured by setting this property.

[default] 500 milliseconds.

8.207.2.16 protocol_overhead_max

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::protocol_overhead_max
```

Maximum size in bytes of protocol overhead, including headers.

This value is the maximum size, in bytes, of protocol-related overhead. Normally, the overhead accounts for UDP and IP headers. The default value is set to accommodate the most common UDP/IP header size.

Note that when **NDDS_Transport_Property_t::message_size_max** (p. 1789) plus this overhead is larger than the UDPv4 maximum message size (65535 bytes), the middleware will automatically reduce the effective message_size↩ _max, to 65535 minus this overhead.

[default] 28.

See also

NDDS_Transport_Property_t::message_size_max (p. 1789)

8.207.2.17 disable_interface_tracking

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::disable_interface_tracking
```

Disables detection of network interface changes.

By default, network interface changes are propagated in the form of locators to other applications. This is done to support IP mobility scenarios.

For example, you could start a RTI Connext application with Wi-Fi and move to a wired connection. In order to continue communicating with other applications, this interface change has to be propagated.

In RTI Connext 5.2 (the initial release) and earlier versions of the product, IP mobility scenarios were not supported. 5.2 applications will report errors if they detect locator changes in a DataWriter or DataReader.

You can disable the notification and propagation of interface changes by setting this property to 1.

This way, an interface change in a newer application will not trigger errors in an application running 5.2 or earlier. Of course, this will prevent the new application from being able to detect network interface changes.

[default] 0

8.207.2.18 public_address

```
char* NDDS_Transport_UDPv4_WAN_Property_t::public_address
```

Public IP address associated with the transport instantiation.

Setting the public IP address is only necessary for the Real-Time WAN Transport associated with an external **dds::domain::DomainParticipant** (p. 1326) (publicly reachable) in order to support the two communication scenarios shown in the Figure below.

For an external **dds::domain::DomainParticipant** (p. 1326) behind a NAT-enabled router, this address is the public IP address of the router.

When this property is set, the DomainParticipant will announce PUBLIC+UUID locators to other DomainParticipants. These locators are reachable locators because they contain this public IP address.

For additional information on Real-Time WAN Transport locators, see the `Core Libraries User's Manual`.

with a Participant that has a Public Address"

[default] null (the transport will announce UUID locators)

8.207.2.19 comm_ports_list

struct **NDDS_Transport_UDP_WAN_CommPortsMappingInfo*** NDDS_Transport_UDPv4_WAN_Property_t::comm_↩ports_list

Configures the public and private UDP ports that a transport instance uses to receive/send RTPS data.

Array containing the mapping between "RTPS ports", "UDP receive host ports", and "UDP receive public ports".

When the transport is configured using properties, the port mapping array is provided using a JSON string.

For example:

```
{
    "default": {"host": 8192, "public": 9678},
    "mappings": [
        {"rtps": 1234, "host": 9999, "public": 5678},
        {"rtps": 1235, "host": 9990, "public": 5679},
    ]
}
```

It is also possible to configure the mapping with XML:

```
<transport_builtin>
    <udpv4_wan>
        <comm_ports>
            <default>
                <host>8192</host>
                <public>9678</public>
            </default>
            <mappings>
                <element>
                    <rtps>1234</rtps>
                    <host>9999</host>
                    <public>5678</public>
                </element>
                <element>
                    <rtps>1235</rtps>
                    <host>9990</host>
                    <public>5679</public>
                </element>
            </mappings>
        </comm_ports>
    </udpv4_wan>
<transport_builtin>
```

For additional information on how to set the value of this property, see the `Core Libraries User's Manual`.

[default] NULL (The UDP ports used for communications will be derived from the RTPS ports associated with the locators for the DomainParticipant and its Endpoints (DataWriters and DataReaders)).

8.207.2.20 comm_ports_list_length

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::comm_ports_list_length
```

Number of elements in the **NDDS_Transport_UDPv4_WAN_Property_t::comm_ports_list** (p. 1816).

[default] 0

8.207.2.21 port_offset

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::port_offset
```

Port offset to allow coexistence with built-in UDPv4 transport.

This property allows using the built-in UDPv4 transport and the Real-Time WAN Transport at the same time.

```
<transport_builtin>

    <mask>UDPv4_WAN|UDPv4</mask>

</transport_builtin>
```

When the UDP ports used by Real-Time WAN Transport are not explicitly set, they are calculated as follows: RTPS port + port_offset.

[default] 125

8.207.2.22 binding_ping_period

```
RTI_UINT32 NDDS_Transport_UDPv4_WAN_Property_t::binding_ping_period
```

Specifies the period in milliseconds at which BINDING PINGS messages are sent to keep NAT mappings open.

Configures the period in milliseconds at which BINDING_PING messages are sent by a local transport instance to a remote transport instance. For example, 1000 means to send BINDING_PING messages every second.

BINDING_PING messages are used on the sending side to open NAT bindings from a local transport instance to a remote transport instance and they are sent periodically to keep the bindings open.

On the receiving side, BINDING_PINGS are used to calculate the public IP transport address of an UUID locator. This address will be used to send data to the locator.

For additional information on the role of BINDING_PING, see the Core Libraries User's Manual.

From a configuration point of view, and to avoid communication disruptions, the period at which a transport instance sends BINDING_PING messages should be smaller than the NAT binding session timeout. This timeout depends on the NAT router configuration.

[default] 1000 (1 sec)

8.208 NDDS_Transport_UDPv6_Property_t Struct Reference

Configurable IPv6/UDP Transport-Plugin properties.

```
#include <transport_udpv6.h>
```

Public Attributes

- struct **NDDS_Transport_Property_t parent**

  Generic properties of all transport plugins.
- RTI_INT32 **send_socket_buffer_size**

  Size in bytes of the send buffer of a socket used for sending.
- RTI_INT32 **recv_socket_buffer_size**

  Size in bytes of the receive buffer of a socket used for receiving.
- RTI_INT32 **unicast_enabled**

  Allows the transport plugin to use unicast for sending and receiving.
- RTI_INT32 **multicast_enabled**

  Allows the transport plugin to use multicast for sending and receiving.
- RTI_INT32 **multicast_ttl**

  Value for the time-to-live parameter for all multicast sends using this plugin.
- RTI_INT32 **multicast_loopback_disabled**

  Prevents the transport plugin from putting multicast packets onto the loopback interface.
- RTI_INT32 **ignore_loopback_interface**

  Prevents the transport plugin from using the IP loopback interface.
- RTI_INT32 **ignore_nonrunning_interfaces**

  Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.
- RTI_INT32 **no_zero_copy**

  ***[DEPRECATED]** Prevents the transport plugin from doing zero copy.*
- RTI_INT32 **send_blocking**

  Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.
- RTI_INT32 **enable_v4mapped**

  Specify whether UDPv6 transport will process IPv4 addresses.
- RTI_UINT32 **transport_priority_mask**

  Set mask for use of transport priority field.
- RTI_INT32 **transport_priority_mapping_low**

  Set low value of output range to IPv6 TCLASS.
- RTI_INT32 **transport_priority_mapping_high**

  Set high value of output range to IPv6 TCLASS.
- RTI_INT32 **send_ping**

  Configures whether to send PING messages.
- RTI_INT32 **force_interface_poll_detection**

  Forces the interface tracker to use a polling mechanism to detect changes on the UDPv6 interfaces.
- RTI_UINT32 **interface_poll_period**

  Specifies the period in milliseconds to query for changes in the state of all the interfaces.
- RTI_INT32 **reuse_multicast_receive_resource**

Controls whether or not to reuse multicast receive resources.

- RTI_INT32 **protocol_overhead_max**

  Maximum size in bytes of protocol overhead, including headers.

- RTI_INT32 **disable_interface_tracking**

  Disables detection of network interface changes.

- RTI_UINT32 **join_multicast_group_timeout**

  [Windows only] Defines how much time (milliseconds) to wait to join a multicast group address when a new interface is detected.

- char * **public_address**

  Public IP address associated with the transport instantiation.

- RTI_INT32 **port_offset**

  *Port offset to allow multiple instances of UDPv6 transports in the same **dds::domain::DomainParticipant** (p. 1326).*

8.208.1 Detailed Description

Configurable IPv6/UDP Transport-Plugin properties.

You can modify the properties in this structure to configure the plugin. However, you must set the properties before the plugin is instantiated.

See also

Transport_Support::set_builtin_transport_property()

8.208.2 Member Data Documentation

8.208.2.1 parent

`struct` **`NDDS_Transport_Property_t`** `NDDS_Transport_UDPv6_Property_t::parent`

Generic properties of all transport plugins.

8.208.2.2 send_socket_buffer_size

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::send_socket_buffer_size
```

Size in bytes of the send buffer of a socket used for sending.

On most operating systems, `setsockopt()` will be called to set the SENDBUF to the value of this parameter.

This value must be greater than or equal to **NDDS_Transport_Property_t::message_size_max** (p. 1789). Its maximum value is dependent on the operating system.

By default, it will be set to UDPv6Transport_SEND_SOCKET_BUFFER_SIZE_DEFAULT.

The maximum value for this property depends on the operating system.

Valid values are:

- UDPv6Transport_SEND_SOCKET_BUFFER_SIZE_DEFAULT: `setsockopt()` (or equivalent) will be called to size the send buffer of the socket with the default value.
- UDPv6Transport_SOCKET_BUFFER_SIZE_OS_DEFAULT: `setsockopt()` (or equivalent) will not be called to size the send buffer of the socket. RTI Connext will use the OS default.
- UDPv6Transport_SOCKET_BUFFER_SIZE_OS_MAX: RTI Connext will retrieve the maximum socket buffer size allowed by the operating system and use that value for the send buffer size of the socket.
- Positive integer: `setsockopt()` (or equivalent) will be called to size the send buffer of the socket with the provided value. If this value exceeds the maximum allowed by the operating system, the operating system's maximum value will be used instead.

8.208.2.3 recv_socket_buffer_size

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::recv_socket_buffer_size
```

Size in bytes of the receive buffer of a socket used for receiving.

On most operating systems, `setsockopt()` will be called to set the RECVBUF to the value of this parameter.

This value must be greater than or equal to **NDDS_Transport_Property_t::message_size_max** (p. 1789). Its maximum value is dependent on the operating system.

By default, it will be set to UDPv6Transport_RECV_SOCKET_BUFFER_SIZE_DEFAULT.

The maximum value for this property depends on the operating system.

Valid values are:

- UDPv6Transport_RECV_SOCKET_BUFFER_SIZE_DEFAULT: `setsockopt()` (or equivalent) will be called to size the receive buffer of the socket with the default value.
- UDPv6Transport_SOCKET_BUFFER_SIZE_OS_DEFAULT: `setsockopt()` (or equivalent) will not be called to size the receive buffer of the socket. RTI Connext will use the OS default.
- UDPv6Transport_SOCKET_BUFFER_SIZE_OS_MAX: RTI Connext will retrieve the maximum socket buffer size allowed by the operating system and use that value for the receive buffer size of the socket.
- Positive integer: `setsockopt()` (or equivalent) will be called to size the receive buffer of the socket with the provided value. If this value exceeds the maximum allowed by the operating system, the operating system's maximum value will be used instead.

8.208.2.4 unicast_enabled

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::unicast_enabled
```

Allows the transport plugin to use unicast for sending and receiving.

This value turns unicast UDP on (if set to 1) or off (if set to 0) for this plugin. By default, it will be turned on (1). Also by default, the plugin will use all the allowed network interfaces that it finds up and running when the plugin is instantiated.

8.208.2.5 multicast_enabled

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::multicast_enabled
```

Allows the transport plugin to use multicast for sending and receiving.

This value turns multicast UDP on (if set to 1) or off (if set to 0) for this plugin. By default, it will be turned on (1) for those platforms that support multicast. Also by default, the plugin will use the all network interfaces allowed for multicast that it finds up and running when the plugin is instantiated.

8.208.2.6 multicast_ttl

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::multicast_ttl
```

Value for the time-to-live parameter for all multicast sends using this plugin.

This is used to set the TTL of multicast packets sent by this transport plugin.

[default] 1

8.208.2.7 multicast_loopback_disabled

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::multicast_loopback_disabled
```

Prevents the transport plugin from putting multicast packets onto the loopback interface.

If multicast loopback is disabled (this value is set to 1), then when sending multicast packets, RTI Connext will *not* put a copy of the packets on the loopback interface. This prevents applications on the same node (including itself) from receiving those packets.

This value is set to 0 by default, meaning multicast loopback is *enabled*.

Disabling multicast loopback (setting this value to 1) may result in minor performance gains when using multicast.

8.208.2.8 ignore_loopback_interface

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::ignore_loopback_interface
```

Prevents the transport plugin from using the IP loopback interface.

Currently three values are allowed:

- **0**: Forces local traffic to be sent over loopback, even if a more efficient transport (such as shared memory) is installed (in which case traffic will be sent over both transports).
- **1**: Disables local traffic via this plugin. Do not use the IP loopback interface even if no NICs are discovered. This is useful when you want applications running on the same node to use a more efficient transport (such as shared memory) instead of the IP loopback.
- **-1**: Automatic. Lets RTI Connext decide between the above two choices.

The current "automatic" (-1) RTI Connext policy is as follows.

- If a shared memory transport plugin is available for local traffic and there is a locator on the initial peers list that can use shared memory, the effective value is 1 (i.e., disable UDPv6 local traffic).
- Otherwise, the effective value is 0 (i.e., use UDPv6 for local traffic also).

[default] -1 Automatic RTI Connext policy based on availability of the shared memory transport.

8.208.2.9 ignore_nonrunning_interfaces

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::ignore_nonrunning_interfaces
```

Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows the same check to be extended to the IFF_↩ RUNNING flag implemented by some operating systems. The RUNNING flag is defined to mean that "all resources are allocated," and may be off if there is no link detected, e.g., the network cable is unplugged.

Two values are allowed:

- **0**: Do not check the RUNNING flag when enumerating interfaces, just make sure interface is UP.
- **1**: Check flag when enumerating interfaces and ignore those that are not reported as RUNNING. This can be used on some operating systems to cause the transport to ignore interfaces that are enabled but not connected to the network.

[default] 1 (i.e., check RUNNING flag)

8.208.2.10 no_zero_copy

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::no_zero_copy
```

[DEPRECATED] Prevents the transport plugin from doing zero copy.

DEPRECATED: This property has no effect. By default, this plugin will use the zero copy on OSes that offer it. While this is good for performance, it may sometimes tax the OS resources in a manner that cannot be overcome by the application.

The best example is if the hardware/device driver lends the buffer to the application itself. If the application does not return the loaned buffers soon enough, the node may error or malfunction. If you cannot reconfigure the H/W, device driver, or the OS to allow the zero copy feature to work for your application, you may have no choice but to turn off the use of zero copy.

By default this is set to 0, so RTI Connext will use the zero copy API if offered by the OS.

8.208.2.11 send_blocking

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::send_blocking
```

Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

Currently two values are defined:

- **NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS:** Sockets are blocking (default socket options for the operating system).
- **NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER:** Sockets are modified to make them non-blocking. THIS MAY CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

[default] NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS.

8.208.2.12 enable_v4mapped

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::enable_v4mapped
```

Specify whether UDPv6 transport will process IPv4 addresses.

Set this to 1 to turn on processing of IPv4 addresses. Note that this may make it incompatible with use of the UDPv4 transport within the same domain participant.

[default] 0.

8.208.2.13 transport_priority_mask

```
RTI_UINT32 NDDS_Transport_UDPv6_Property_t::transport_priority_mask
```

Set mask for use of transport priority field.

If transport priority mapping is supported on the platform, this mask is used in conjunction with **NDDS_↩ Transport_UDPv6_Property_t::transport_priority_mapping_low** (p. 1825) and **NDDS_Transport_UDPv6_↩ Property_t::transport_priority_mapping_high** (p. 1825) to define the mapping from the DDS transport priority (see **TRANSPORT_PRIORITY** (p. 550)) to the IPv6 TCLASS field. Defines a contiguous region of bits in the 32-bit transport priority value that is used to generate values for the IPv6 TCLASS field on an outgoing socket.

For example, the value 0x0000ff00 causes bits 9-16 (8 bits) to be used in the mapping. The value will be scaled from the mask range (0x0000 - 0xff00 in this case) to the range specified by low and high.

If the mask is set to zero, then the transport will not set IPv6 TCLASS for send sockets.

[default] 0.

8.208.2.14 transport_priority_mapping_low

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_low
```

Set low value of output range to IPv6 TCLASS.

This is used in conjunction with **NDDS_Transport_UDPv6_Property_t::transport_priority_mask** (p. 1824) and **NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_high** (p. 1825) to define the mapping from the DDS transport priority to the IPv6 TCLASS field. Defines the low value of the output range for scaling.

Note that IPv6 TCLASS is generally an 8-bit value.

[default] 0.

8.208.2.15 transport_priority_mapping_high

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_high
```

Set high value of output range to IPv6 TCLASS.

This is used in conjunction with **NDDS_Transport_UDPv6_Property_t::transport_priority_mask** (p. 1824) and **NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_low** (p. 1825) to define the mapping from the DDS transport priority to the IPv6 TCLASS field. Defines the high value of the output range for scaling.

Note that IPv6 TCLASS is generally an 8-bit value.

[default] 0xff.

8.208.2.16 send_ping

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::send_ping
```

Configures whether to send PING messages.

See also

NDDS_Transport_UDPv4_Property_t::send_ping (p. 1805)

8.208.2.17 force_interface_poll_detection

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::force_interface_poll_detection
```

Forces the interface tracker to use a polling mechanism to detect changes on the UDPv6 interfaces.

See also

NDDS_Transport_UDPv4_Property_t::force_interface_poll_detection (p. 1806)

8.208.2.18 interface_poll_period

```
RTI_UINT32 NDDS_Transport_UDPv6_Property_t::interface_poll_period
```

Specifies the period in milliseconds to query for changes in the state of all the interfaces.

See also

NDDS_Transport_UDPv4_Property_t::interface_poll_period (p. 1806)

8.208.2.19 reuse_multicast_receive_resource

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::reuse_multicast_receive_resource
```

Controls whether or not to reuse multicast receive resources.

See also

NDDS_Transport_UDPv4_Property_t::reuse_multicast_receive_resource (p. 1806)

8.208.2.20 protocol_overhead_max

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::protocol_overhead_max
```

Maximum size in bytes of protocol overhead, including headers.

This value is the maximum size, in bytes, of protocol-related overhead. Normally, the overhead accounts for UDP and IP headers. The default value is set to accommodate the most common UDP/IP header size.

Note that when **NDDS_Transport_Property_t::message_size_max** (p. 1789) plus this overhead is larger than the UDPv6 maximum message size (65535 bytes), the middleware will automatically reduce the effective message_size↩ _max, to 65535 minus this overhead.

[default] 48.

See also

NDDS_Transport_Property_t::message_size_max (p. 1789)

8.208.2.21 disable_interface_tracking

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::disable_interface_tracking
```

Disables detection of network interface changes.

See also

NDDS_Transport_UDPv4_Property_t::disable_interface_tracking (p. 1807)

8.208.2.22 join_multicast_group_timeout

```
RTI_UINT32 NDDS_Transport_UDPv6_Property_t::join_multicast_group_timeout
```

[Windows only] Defines how much time (milliseconds) to wait to join a multicast group address when a new interface is detected.

See also

NDDS_Transport_UDPv4_Property_t::join_multicast_group_timeout (p. 1807)

8.208.2.23 public_address

```
char* NDDS_Transport_UDPv6_Property_t::public_address
```

Public IP address associated with the transport instantiation.

Setting the public IP address is only necessary to support communication over WAN that involves Network Address Translation (NAT).

Typically, the address is the public address of the IP NAT router that provides access to the WAN.

By default, the **dds::domain::DomainParticipant** (p. 1326) creating the transport will announce the IP addresses obtained from the NICs to other DomainParticipants in the system.

When this property is set, the DomainParticipant will announce the IP address corresponding to the property value instead of the LAN IP addresses associated with the NICs.

Note 1: Setting this property is necessary, but is not a sufficient condition for sending and receiving data over the WAN. You must also configure the IP NAT router to allow UDP traffic and to map the public IP address specified by this property to the DomainParticipant's private LAN IP address. This is typically done with one of the following mechanisms:

- Port Forwarding: You must map the private ports used to receive discovery and user data traffic to the corresponding public ports (see **rti::core::RtpsWellKnownPorts** (p. 2258)). Public and private ports must be the same since the transport does not allow you to change the mapping.
- 1:1 NAT: You must add a 1:1 NAT entry that maps the public IP address specified in this property to the private LAN IP address of the DomainParticipant.

Note 2: By setting this property, the **dds::domain::DomainParticipant** (p. 1326) only announces its public IP address to other DomainParticipants. Therefore, communication with DomainParticipants within the LAN that are running on different nodes will not work unless the NAT router is configured to enable NAT reflection (hairpin NAT).

There is another way to achieve simultaneous communication with DomainParticipants running in the LAN and WAN, that does not require hairpin NAT. This way uses a gateway application such as RTI Routing Service to provide access to the WAN.

[default] null (the transport uses the IP addresses obtained from the NICs)

8.208.2.24 port_offset

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::port_offset
```

Port offset to allow multiple instances of UDPv6 transports in the same **dds::domain::DomainParticipant** (p. 1326).

By default, it is not possible to have multiple instances of the UDPv6 transport in the same **dds::domain::Domain↩ Participant** (p. 1326). This is because all instances will try to bind to the same UDP port(s) for receiving data.

With this property, you can specify an offset that will be added to the RTPS port(s) to determine the UDP port(s) to bind to. This way, you can have multiple instances of the UDPv6 transport in the same **dds::domain::DomainParticipant** (p. 1326).

An RTPS port will be mapped to an UDP port as follows: UDP port = RTPS port + port_offset.

[default] 0

8.209 NDDS_Transport_UUID Struct Reference

Unequivocally identifies a transport plugin instance.

`#include <transport_common.h>`

8.209.1 Detailed Description

Unequivocally identifies a transport plugin instance.

8.210 rti::util::network_capture::NetworkCaptureParams Class Reference

<<***extension***>> *(*p. *354)* Input parameters for starting Network Capture.

`#include <network_capture.hpp>`

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **dds::core::StringSeq transports** () const
  Get the list of transports to capture.
- **NetworkCaptureParams** & **transports** (const **dds::core::StringSeq** &transports_in)
  Set the list of transports to capture.
- **ContentKindMask dropped_content** () const
  Get the type of contents to exclude from the capture file.
- **NetworkCaptureParams** & **dropped_content** (**ContentKindMask** dropped_content_in)
  Set the type of contents to exclude from the capture file.
- **TrafficKindMask traffic** () const
  Get the traffic direction to capture.
- **NetworkCaptureParams** & **traffic** (**TrafficKindMask** traffic_in)
  Set the traffic direction to capture.
- bool **parse_encrypted_content** () const
  Get whether Network Capture will parse encrypted contents.
- **NetworkCaptureParams** & **parse_encrypted_content** (bool parse_encrypted_content_in)
  Set whether Network Capture will parse encrypted contents.
- const **rti::core::ThreadSettings** & **checkpoint_thread_settings** () const
  Get the properties of the checkpoint thread.
- **NetworkCaptureParams** & **checkpoint_thread_settings** (const **rti::core::ThreadSettings** &checkpoint_↩ thread_settings_in)
  Set the properties of the checkpoint thread.
- int32_t **frame_queue_size** () const
  Get the size of the frame queue (Bytes).
- **NetworkCaptureParams** & **frame_queue_size** (int32_t frame_queue_size_in)
  Set the size of the frame queue.

8.210.1 Detailed Description

<<***extension***>> *(p. 354)* Input parameters for starting Network Capture.

8.210.2 Member Function Documentation

8.210.2.1 transports() [1/2]

```
dds::core::StringSeq rti::util::network_capture::NetworkCaptureParams::transports ( ) const
```

Get the list of transports to capture.

Returns

List of transports to capture.

8.210.2.2 transports() [2/2]

```
NetworkCaptureParams & rti::util::network_capture::NetworkCaptureParams::transports (
            const  dds::core::StringSeq & transports_in )
```

Set the list of transports to capture.

Network Capture will only save RTPS frames if the associated transport protocol is part of this sequence.

[default] Empty sequence initializer. This means that by default all transports will be captured.

Parameters

transports_in	The sequence of transports that we want to capture traffic from.

Returns

The Network Capture's parameter instance.

8.210.2.3 dropped_content() [1/2]

```
ContentKindMask rti::util::network_capture::NetworkCaptureParams::dropped_content ( ) const
```

Get the type of contents to exclude from the capture file.

Returns

Contents to exclude from the capture file.

8.210.2.4 dropped_content() [2/2]

```
NetworkCaptureParams & rti::util::network_capture::NetworkCaptureParams::dropped_content (
            ContentKindMask dropped_content_in )
```

Set the type of contents to exclude from the capture file.

It accepts values from **network_capture::ContentKindMask** (p. 962) .

We can choose to exclude user data or encrypted content from the capture file.

[default] No content is excluded - **network_capture::ContentKindMask::default_mask()** (p. 964).

Parameters

dropped_content_in	The mask of contents to exclude from the capture.

Returns

The Network Capture's parameter instance.

8.210.2.5 traffic() [1/2]

```
TrafficKindMask rti::util::network_capture::NetworkCaptureParams::traffic ( ) const
```

Get the traffic direction to capture.

Returns

Traffic direction to capture.

8.210.2.6 traffic() [2/2]

```
NetworkCaptureParams & rti::util::network_capture::NetworkCaptureParams::traffic (
            TrafficKindMask traffic_in )
```

Set the traffic direction to capture.

It accepts values from **network_capture::TrafficKindMask** (p. 2529) .

[default] Capture both inbound and outbound traffic - **network_capture::TrafficKindMask::default_mask()** (p. 2532).

Parameters

traffic↩ _in	The mask with the traffic directions that we want to capture.

Returns

The Network Capture's parameter instance.

8.210.2.7 parse_encrypted_content() [1/2]

```
bool rti::util::network_capture::NetworkCaptureParams::parse_encrypted_content ( ) const
```

Get whether Network Capture will parse encrypted contents.

Returns

Whether Network Capture will parse encrypted contents.

8.210.2.8 parse_encrypted_content() [2/2]

```
NetworkCaptureParams & rti::util::network_capture::NetworkCaptureParams::parse_encrypted_content
(
            bool parse_encrypted_content_in )
```

Set whether Network Capture will parse encrypted contents.

Network Capture supports decryption of RTPS messages and submessages. It does not support decryption of the user data payload (<data_protection_kind> tag in the GovernanceDocument).

[default] false

Parameters

parse_encrypted_content_in	Boolean indicating whether to parse or not encrypted content.

Returns

The Network Capture's parameter instance.

8.210.2.9 checkpoint_thread_settings() [1/2]

```
const rti::core::ThreadSettings & rti::util::network_capture::NetworkCaptureParams::checkpoint_thread_settings ( ) const
```

Get the properties of the checkpoint thread.

Returns

Properties of the checkpoint thread.

8.210.2.10 checkpoint_thread_settings() [2/2]

```
NetworkCaptureParams & rti::util::network_capture::NetworkCaptureParams::checkpoint_thread_settings (
            const rti::core::ThreadSettings & checkpoint_thread_settings_in )
```

Set the properties of the checkpoint thread.

The checkpoint thread is a per-participant thread responsible for reading frames from a queue and saving them to disk (as soon as they are produced).

The members that can be configured are:

- **rti::core::ThreadSettings::mask** (p. 2450). Default value of (rti::core::ThreadSettingsKindMask::THREAD_SETTINGS_PRIORITY_ENFORCE | **rti::core::ThreadSettingsKindMask::stdio()** (p. 2456)).
- **rti::core::ThreadSettings::priority** (p. 2450). Platform-dependent - Consult `Platform Notes` for additional details.
- **rti::core::ThreadSettings::stack_size** (p. 2451). Platform-dependent - Consult `Platform Notes` for additional details.

Parameters

checkpoint_thread_settings_in	Properties of the checkpoint thread that we want to set.

Returns

The Network Capture's parameter instance.

8.210.2.11 frame_queue_size() [1/2]

```
int32_t rti::util::network_capture::NetworkCaptureParams::frame_queue_size ( ) const
```

Get the size of the frame queue (Bytes).

Network Capture enqueues frames before saving them to disk, which takes place in a separate thread.

Network Capture does not block if the queue becomes full. Attempting to enqueue a frame with a full frame queue will fail (frame won't be captured) with a log message.

The size of the queue is dependent on the network traffic (amount of frames that we want to capture) and system resources (how fast we can capture frames).

[default] 2097152 (2MB).

Returns

Size of the frame queue (Bytes).

8.210.2.12 frame_queue_size() [2/2]

```
NetworkCaptureParams & rti::util::network_capture::NetworkCaptureParams::frame_queue_size (
            int32_t frame_queue_size_in )
```

Set the size of the frame queue.

Network Capture enqueues frames before saving them to disk, which takes place in a separate thread.

Network Capture does not block if the queue becomes full. Attempting to enqueue a frame with a full frame queue will fail (frame won't be captured) with a log message.

The size of the queue is dependent on the network traffic (amount of frames that we want to capture) and system resources (how fast we can capture frames).

[default] 2097152 (2MB).

Parameters

frame_queue_size_in	Size of the frame queue in Bytes.

Returns

The Network Capture's parameter instance.

8.211 rti::topic::no_compile_data_t Struct Reference

The type to specify as the CompileData template parameter to your **ContentFilter** (p. 950) if your compile function does not return any data.

```
#include <rti/topic/ContentFilter.hpp>
```

8.211.1 Detailed Description

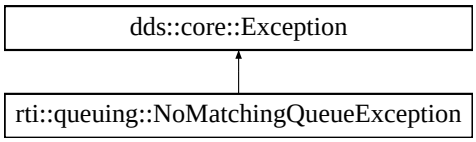
The type to specify as the CompileData template parameter to your **ContentFilter** (p. 950) if your compile function does not return any data.

8.212 rti::queuing::NoMatchingQueueException Class Reference

The entity lost matching with all the Queuing Service.

```
#include <QueueSupport.hpp>
```

Inheritance diagram for rti::queuing::NoMatchingQueueException:



Public Member Functions

- const char * **what** () const throw ()

  *Access the message contained in this **NoMatchingQueueException** (p. 1835) exception.*

8.212.1 Detailed Description

The entity lost matching with all the Queuing Service.

Only operations related with sending samples or waiting for acknowledgements can throw this exception.

8.212.2 Member Function Documentation

8.212.2.1 what()

```
const char * rti::queuing::NoMatchingQueueException::what ( ) const throw ( )   [inline], [virtual]
```

Access the message contained in this **NoMatchingQueueException** (p. 1835) exception.

Returns

The message.

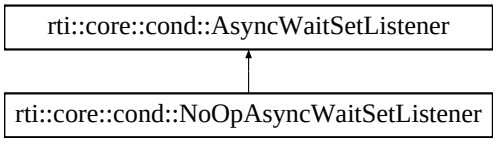
Implements **dds::core::Exception** (p. 1547).

8.213 rti::core::cond::NoOpAsyncWaitSetListener Class Reference

A convenience implementation of **AsyncWaitSetListener** (p. 856) where all methods are overridden to do nothing.

```
#include <AsyncWaitSetListener.hpp>
```

Inheritance diagram for rti::core::cond::NoOpAsyncWaitSetListener:



Public Member Functions

- virtual void **on_thread_spawned** (ThreadId)
  No-op.
- virtual void **on_thread_deleted** (ThreadId)
  No-op.
- virtual void **on_wait_timeout** (ThreadId)
  No-op.

8.213.1 Detailed Description

A convenience implementation of **AsyncWaitSetListener** (p. 856) where all methods are overridden to do nothing.

Most of the time, you can derive your listener from this class so you don't have to implement the methods you don't need.

8.213.2 Member Function Documentation

8.213.2.1 on_thread_spawned()

```
virtual void rti::core::cond::NoOpAsyncWaitSetListener::on_thread_spawned (
            ThreadId  )  [inline], [virtual]
```

No-op.

Implements **rti::core::cond::AsyncWaitSetListener** (p. 857).

8.213.2.2 on_thread_deleted()

```
virtual void rti::core::cond::NoOpAsyncWaitSetListener::on_thread_deleted (
            ThreadId  )  [inline], [virtual]
```

No-op.

Implements **rti::core::cond::AsyncWaitSetListener** (p. 857).

8.213.2.3 on_wait_timeout()

```
virtual void rti::core::cond::NoOpAsyncWaitSetListener::on_wait_timeout (
            ThreadId  )  [inline], [virtual]
```

No-op.

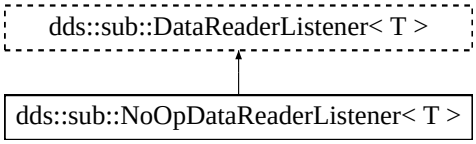
Implements **rti::core::cond::AsyncWaitSetListener** (p. 857).

8.214 dds::sub::NoOpDataReaderListener< T > Class Template Reference

A convenience implementation of **DataReaderListener** (p. 1055) where all methods are overridden to do nothing.

`#include "dds/sub/DataReaderListener.hpp"`

Inheritance diagram for dds::sub::NoOpDataReaderListener< T >:



Public Member Functions

- virtual void **on_requested_deadline_missed** (**DataReader**< T > &, const **dds::core::status::Requested↩ DeadlineMissedStatus** &)

  No-op.
- virtual void **on_requested_incompatible_qos** (**DataReader**< T > &, const **dds::core::status::Requested↩ IncompatibleQosStatus** &)

  No-op.
- virtual void **on_sample_rejected** (**DataReader**< T > &, const **dds::core::status::SampleRejectedStatus** &)

  No-op.
- virtual void **on_liveliness_changed** (**DataReader**< T > &, const **dds::core::status::LivelinessChanged↩ Status** &)

  No-op.
- virtual void **on_data_available** (**DataReader**< T > &)

  No-op.
- virtual void **on_subscription_matched** (**DataReader**< T > &, const **dds::core::status::Subscription↩ MatchedStatus** &)

  No-op.
- virtual void **on_sample_lost** (**DataReader**< T > &, const **dds::core::status::SampleLostStatus** &)

  No-op.

8.214.1 Detailed Description

template<**typename T**>
class dds::sub::NoOpDataReaderListener< **T** >

A convenience implementation of **DataReaderListener** (p. 1055) where all methods are overridden to do nothing.

Most of the time, you can derive your listener from this class so you don't have to implement the methods you don't need.

8.214.2 Member Function Documentation

8.214.2.1 on_requested_deadline_missed()

```
template<typename T >
virtual void  dds::sub::NoOpDataReaderListener< T >::on_requested_deadline_missed (
              DataReader< T > & ,
             const  dds::core::status::RequestedDeadlineMissedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::DataReaderListener< T >** (p. 1057).

8.214.2.2 on_requested_incompatible_qos()

```
template<typename T >
virtual void  dds::sub::NoOpDataReaderListener< T >::on_requested_incompatible_qos (
              DataReader< T > & ,
             const  dds::core::status::RequestedIncompatibleQosStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::DataReaderListener< T >** (p. 1057).

8.214.2.3 on_sample_rejected()

```
template<typename T >
virtual void  dds::sub::NoOpDataReaderListener< T >::on_sample_rejected (
              DataReader< T > & ,
             const  dds::core::status::SampleRejectedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::DataReaderListener< T >** (p. 1057).

8.214.2.4 on_liveliness_changed()

```
template<typename T >
virtual void  dds::sub::NoOpDataReaderListener< T >::on_liveliness_changed (
              DataReader< T > & ,
             const  dds::core::status::LivelinessChangedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::DataReaderListener< T >** (p. 1057).

8.214.2.5 on_data_available()

```
template<typename T >
virtual void  dds::sub::NoOpDataReaderListener< T >::on_data_available (
              DataReader< T > &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::DataReaderListener< T >** (p. 1058).

8.214.2.6 on_subscription_matched()

```
template<typename T >
virtual void  dds::sub::NoOpDataReaderListener< T >::on_subscription_matched (
              DataReader< T > & ,
             const  dds::core::status::SubscriptionMatchedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::DataReaderListener< T >** (p. 1058).

8.214.2.7 on_sample_lost()

```
template<typename T >
virtual void  dds::sub::NoOpDataReaderListener< T >::on_sample_lost (
              DataReader< T > & ,
             const  dds::core::status::SampleLostStatus &  )  [inline], [virtual]
```

No-op.

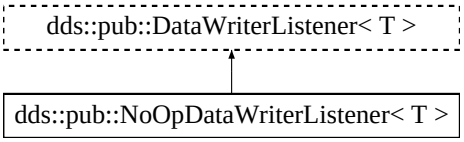
Implements **dds::sub::DataReaderListener< T >** (p. 1058).

8.215 dds::pub::NoOpDataWriterListener< T > Class Template Reference

A convenience implementation of **DataWriterListener** (p. 1202) where all methods are overridden to do nothing.

`#include "dds/pub/DataWriterListener.hpp"`

Inheritance diagram for dds::pub::NoOpDataWriterListener< T >:



Public Member Functions

- virtual void **on_offered_deadline_missed** (**dds::pub::DataWriter**< T > &, const **dds::core::status::OfferedDeadlineMissedStatus** &)

  No-op.
- virtual void **on_offered_incompatible_qos** (**dds::pub::DataWriter**< T > &, const **dds::core::status::OfferedIncompatibleQosStatus** &)

  No-op.
- virtual void **on_liveliness_lost** (**dds::pub::DataWriter**< T > &, const **dds::core::status::LivelinessLostStatus** &)

  No-op.
- virtual void **on_publication_matched** (**dds::pub::DataWriter**< T > &, const **dds::core::status::PublicationMatchedStatus** &)

  No-op.
- virtual void **on_reliable_writer_cache_changed** (**dds::pub::DataWriter**< T > &, const **rti::core::status::ReliableWriterCacheChangedStatus** &)

  <<***extension***>> *(p. 354) No-op*
- virtual void **on_reliable_reader_activity_changed** (**dds::pub::DataWriter**< T > &, const **rti::core::status::ReliableReaderActivityChangedStatus** &)

  <<***extension***>> *(p. 354) No-op*
- virtual void **on_instance_replaced** (**dds::pub::DataWriter**< T > &, const **dds::core::InstanceHandle** &)

  <<***extension***>> *(p. 354) No-op*
- virtual void **on_application_acknowledgment** (**dds::pub::DataWriter**< T > &, const **rti::pub::AcknowledgmentInfo** &)

  <<***extension***>> *(p. 354) No-op*
- virtual void **on_service_request_accepted** (**dds::pub::DataWriter**< T > &, const **rti::core::status::ServiceRequestAcceptedStatus** &)

  <<***extension***>> *(p. 354) No-op*
- virtual void **on_destination_unreachable** (**dds::pub::DataWriter**< T > &, const **dds::core::InstanceHandle** &, const **rti::core::Locator** &)

  <<***extension***>> *(p. 354) No-op*
- virtual void * **on_data_request** (**dds::pub::DataWriter**< T > &, const **rti::core::Cookie** &)

  <<***extension***>> *(p. 354) No-op*
- virtual void **on_data_return** (**dds::pub::DataWriter**< T > &, void *, const **rti::core::Cookie** &)

  <<***extension***>> *(p. 354) No-op*
- virtual void **on_sample_removed** (**dds::pub::DataWriter**< T > &, const **rti::core::Cookie** &)

  <<***extension***>> *(p. 354) No-op*

8.215.1 Detailed Description

template<**typename T**>
class dds::pub::NoOpDataWriterListener< **T** >

A convenience implementation of **DataWriterListener** (p. 1202) where all methods are overridden to do nothing.

Most of the time, you can derive your listener from this class so you don't have to implement the methods you don't need.

8.215.2 Member Function Documentation

8.215.2.1 on_offered_deadline_missed()

```
template<typename T >
virtual void  dds::pub::NoOpDataWriterListener< T >::on_offered_deadline_missed (
              dds::pub::DataWriter< T > & ,
             const  dds::core::status::OfferedDeadlineMissedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::DataWriterListener**< **T** > (p. 1203).

8.215.2.2 on_offered_incompatible_qos()

```
template<typename T >
virtual void  dds::pub::NoOpDataWriterListener< T >::on_offered_incompatible_qos (
              dds::pub::DataWriter< T > & ,
             const  dds::core::status::OfferedIncompatibleQosStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::DataWriterListener**< **T** > (p. 1204).

8.215.2.3 on_liveliness_lost()

```
template<typename T >
virtual void  dds::pub::NoOpDataWriterListener< T >::on_liveliness_lost (
              dds::pub::DataWriter< T > & ,
             const  dds::core::status::LivelinessLostStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::DataWriterListener**< **T** > (p. 1204).

8.215.2.4 on_publication_matched()

```
template<typename T >
virtual void  dds::pub::NoOpDataWriterListener< T >::on_publication_matched (
              dds::pub::DataWriter< T > & ,
             const  dds::core::status::PublicationMatchedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::DataWriterListener< T >** (p. 1205).

8.215.2.5 on_reliable_writer_cache_changed()

```
template<typename T >
virtual void  dds::pub::NoOpDataWriterListener< T >::on_reliable_writer_cache_changed (
              dds::pub::DataWriter< T > & ,
             const  rti::core::status::ReliableWriterCacheChangedStatus &  )  [inline], [virtual]
```

<<***extension***>> *(p. 354)* No-op

Implements **dds::pub::DataWriterListener< T >** (p. 1205).

8.215.2.6 on_reliable_reader_activity_changed()

```
template<typename T >
virtual void  dds::pub::NoOpDataWriterListener< T >::on_reliable_reader_activity_changed (
              dds::pub::DataWriter< T > & ,
             const  rti::core::status::ReliableReaderActivityChangedStatus &  )  [inline], [virtual]
```

<<***extension***>> *(p. 354)* No-op

Implements **dds::pub::DataWriterListener< T >** (p. 1206).

8.215.2.7 on_instance_replaced()

```
template<typename T >
virtual void  dds::pub::NoOpDataWriterListener< T >::on_instance_replaced (
              dds::pub::DataWriter< T > & ,
             const  dds::core::InstanceHandle &  )  [inline], [virtual]
```

<<***extension***>> *(p. 354)* No-op

Implements **dds::pub::DataWriterListener< T >** (p. 1206).

8.215.2.8 on_application_acknowledgment()

```
template<typename T >
virtual void  dds::pub::NoOpDataWriterListener< T >::on_application_acknowledgment (
              dds::pub::DataWriter< T > & ,
             const  rti::pub::AcknowledgmentInfo &  )  [inline], [virtual]
```

<<***extension***>> *(*p. *354)* No-op

Implements **dds::pub::DataWriterListener< T >** (p. 1207).

8.215.2.9 on_service_request_accepted()

```
template<typename T >
virtual void  dds::pub::NoOpDataWriterListener< T >::on_service_request_accepted (
              dds::pub::DataWriter< T > & ,
             const  rti::core::status::ServiceRequestAcceptedStatus &  )  [inline], [virtual]
```

<<***extension***>> *(*p. *354)* No-op

Implements **dds::pub::DataWriterListener< T >** (p. 1209).

8.215.2.10 on_destination_unreachable()

```
template<typename T >
virtual void  dds::pub::NoOpDataWriterListener< T >::on_destination_unreachable (
              dds::pub::DataWriter< T > & ,
             const  dds::core::InstanceHandle & ,
             const  rti::core::Locator &  )  [inline], [virtual]
```

<<***extension***>> *(*p. *354)* No-op

Implements **dds::pub::DataWriterListener< T >** (p. 1202).

8.215.2.11 on_data_request()

```
template<typename T >
virtual void *  dds::pub::NoOpDataWriterListener< T >::on_data_request (
              dds::pub::DataWriter< T > & ,
             const  rti::core::Cookie &  )  [inline], [virtual]
```

<<***extension***>> *(*p. *354)* No-op

Implements **dds::pub::DataWriterListener< T >** (p. 1202).

8.215.2.12 on_data_return()

```
template<typename T >
virtual void  dds::pub::NoOpDataWriterListener< T >::on_data_return (
              dds::pub::DataWriter< T > & ,
             void * ,
             const  rti::core::Cookie &  )  [inline], [virtual]
```

<<***extension***>> *(*p. *354)* No-op

Implements **dds::pub::DataWriterListener**< **T** > (p. 1202).

8.215.2.13 on_sample_removed()

```
template<typename T >
virtual void  dds::pub::NoOpDataWriterListener< T >::on_sample_removed (
              dds::pub::DataWriter< T > & ,
             const  rti::core::Cookie &  )  [inline], [virtual]
```

<<***extension***>> *(*p. *354)* No-op

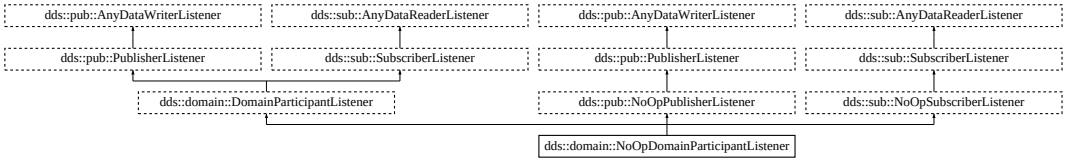
Implements **dds::pub::DataWriterListener**< **T** > (p. 1209).

8.216 dds::domain::NoOpDomainParticipantListener Class Reference

A convenience implementation of **DomainParticipantListener** (p. 1382) where all methods are overridden to do nothing.

```
#include "dds/domain/DomainParticipantListener.hpp"
```

Inheritance diagram for dds::domain::NoOpDomainParticipantListener:



Public Member Functions

- virtual void **on_offered_deadline_missed** (**dds::pub::AnyDataWriter** &, const **::dds::core::status::↩ OfferedDeadlineMissedStatus** &)

  No-op.
- virtual void **on_offered_incompatible_qos** (**dds::pub::AnyDataWriter** &, const **::dds::core::status::↩ OfferedIncompatibleQosStatus** &)

  No-op.
- virtual void **on_liveliness_lost** (**dds::pub::AnyDataWriter** &, const **::dds::core::status::LivelinessLost↩ Status** &)

  No-op.
- virtual void **on_publication_matched** (**dds::pub::AnyDataWriter** &, const **::dds::core::status::Publication↩ MatchedStatus** &)

  No-op.
- virtual void **on_reliable_writer_cache_changed** (**dds::pub::AnyDataWriter** &, const **rti::core::status::↩ ReliableWriterCacheChangedStatus** &)

  <<***extension***>> *(*p. *354) No-op*
- virtual void **on_reliable_reader_activity_changed** (**dds::pub::AnyDataWriter** &, const **rti::core::status::↩ ReliableReaderActivityChangedStatus** &)

  <<***extension***>> *(*p. *354) No-op*
- virtual void **on_sample_removed** (**dds::pub::AnyDataWriter** &, const **rti::core::Cookie** &)

  <<***extension***>> *(*p. *354) No-op*
- virtual void **on_instance_replaced** (**dds::pub::AnyDataWriter** &, const **dds::core::InstanceHandle** &)

  <<***extension***>> *(*p. *354) No-op*
- virtual void **on_application_acknowledgment** (**dds::pub::AnyDataWriter** &, const **rti::pub::↩ AcknowledgmentInfo** &)

  <<***extension***>> *(*p. *354) No-op*
- virtual void **on_service_request_accepted** (**dds::pub::AnyDataWriter** &, const **rti::core::status::Service↩ RequestAcceptedStatus** &)

  <<***extension***>> *(*p. *354) No-op*
- virtual void **on_invalid_local_status_advance_notice** (**DomainParticipant** &, const **rti::core::status::↩ InvalidLocalAdvanceNoticeStatus** &)

  <<***extension***>> *(*p. *354) No-op*
- virtual void **on_data_on_readers** (**dds::sub::Subscriber** &)

  No-op.
- virtual void **on_requested_deadline_missed** (**dds::sub::AnyDataReader** &, const **dds::core::status::↩ RequestedDeadlineMissedStatus** &)

  No-op.
- virtual void **on_requested_incompatible_qos** (**dds::sub::AnyDataReader** &, const **dds::core::status::↩ RequestedIncompatibleQosStatus** &)

  No-op.
- virtual void **on_sample_rejected** (**dds::sub::AnyDataReader** &, const **dds::core::status::SampleRejected↩ Status** &)

  No-op.
- virtual void **on_liveliness_changed** (**dds::sub::AnyDataReader** &, const **dds::core::status::Liveliness↩ ChangedStatus** &)

  No-op.
- virtual void **on_data_available** (**dds::sub::AnyDataReader** &)

  No-op.

- virtual void **on_subscription_matched** (**dds::sub::AnyDataReader** &, const **dds::core::status::SubscriptionMatchedStatus** &)

  No-op.
- virtual void **on_sample_lost** (**dds::sub::AnyDataReader** &, const **dds::core::status::SampleLostStatus** &)

  No-op.
- virtual void **on_inconsistent_topic** (**dds::topic::AnyTopic** &, const **dds::core::status::InconsistentTopicStatus** &)

  No-op.

8.216.1 Detailed Description

A convenience implementation of **DomainParticipantListener** (p. 1382) where all methods are overridden to do nothing.

Most of the time, you can derive your listener from this class so you don't have to implement the methods you don't need.

8.216.2 Member Function Documentation

8.216.2.1 on_offered_deadline_missed()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_offered_deadline_missed (
              dds::pub::AnyDataWriter & ,
             const  ::dds::core::status::OfferedDeadlineMissedStatus &  )  [inline], [virtual]
```

No-op.

Reimplemented from **dds::pub::NoOpPublisherListener** (p. 1853).

8.216.2.2 on_offered_incompatible_qos()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_offered_incompatible_qos (
              dds::pub::AnyDataWriter & ,
             const  ::dds::core::status::OfferedIncompatibleQosStatus &  )  [inline], [virtual]
```

No-op.

Reimplemented from **dds::pub::NoOpPublisherListener** (p. 1854).

8.216.2.3 on_liveliness_lost()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_liveliness_lost (
             dds::pub::AnyDataWriter & ,
            const  ::dds::core::status::LivelinessLostStatus &  )  [inline], [virtual]
```

No-op.

Reimplemented from **dds::pub::NoOpPublisherListener** (p. 1854).

8.216.2.4 on_publication_matched()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_publication_matched (
             dds::pub::AnyDataWriter & ,
            const  ::dds::core::status::PublicationMatchedStatus &  )  [inline], [virtual]
```

No-op.

Reimplemented from **dds::pub::NoOpPublisherListener** (p. 1854).

8.216.2.5 on_reliable_writer_cache_changed()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_reliable_writer_cache_changed (
             dds::pub::AnyDataWriter & ,
            const  rti::core::status::ReliableWriterCacheChangedStatus &  )  [inline], [virtual]
```

<<***extension***>> *(p. 354)* No-op

Reimplemented from **dds::pub::NoOpPublisherListener** (p. 1854).

8.216.2.6 on_reliable_reader_activity_changed()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_reliable_reader_activity_changed (
             dds::pub::AnyDataWriter & ,
            const  rti::core::status::ReliableReaderActivityChangedStatus &  )  [inline], [virtual]
```

<<***extension***>> *(p. 354)* No-op

Reimplemented from **dds::pub::NoOpPublisherListener** (p. 1855).

8.216.2.7 on_sample_removed()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_sample_removed (
             dds::pub::AnyDataWriter & ,
            const  rti::core::Cookie &  )  [inline], [virtual]
```

<<***extension***>> *(p. 354)* No-op

Reimplemented from **dds::pub::NoOpPublisherListener** (p. 1855).

8.216.2.8 on_instance_replaced()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_instance_replaced (
             dds::pub::AnyDataWriter & ,
            const  dds::core::InstanceHandle &  )  [inline], [virtual]
```

<<***extension***>> *(p. 354)* No-op

Reimplemented from **dds::pub::NoOpPublisherListener** (p. 1855).

8.216.2.9 on_application_acknowledgment()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_application_acknowledgment (
             dds::pub::AnyDataWriter & ,
            const  rti::pub::AcknowledgmentInfo &  )  [inline], [virtual]
```

<<***extension***>> *(p. 354)* No-op

Reimplemented from **dds::pub::NoOpPublisherListener** (p. 1855).

8.216.2.10 on_service_request_accepted()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_service_request_accepted (
             dds::pub::AnyDataWriter & ,
            const  rti::core::status::ServiceRequestAcceptedStatus &  )  [inline], [virtual]
```

<<***extension***>> *(p. 354)* No-op

Reimplemented from **dds::pub::NoOpPublisherListener** (p. 1856).

8.216.2.11 on_invalid_local_status_advance_notice()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_invalid_local_status_advance_notice (
             DomainParticipant & ,
            const  rti::core::status::InvalidLocalAdvanceNoticeStatus &  )  [inline], [virtual]
```

<<***extension***>> *(*p. *354)* No-op

Implements **dds::domain::DomainParticipantListener** (p. 1382).

8.216.2.12 on_data_on_readers()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_data_on_readers (
             dds::sub::Subscriber &  )  [inline], [virtual]
```

No-op.

Reimplemented from **dds::sub::NoOpSubscriberListener** (p. 1865).

8.216.2.13 on_requested_deadline_missed()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_requested_deadline_missed (
             dds::sub::AnyDataReader & ,
            const  dds::core::status::RequestedDeadlineMissedStatus &  )  [inline], [virtual]
```

No-op.

Reimplemented from **dds::sub::NoOpSubscriberListener** (p. 1865).

8.216.2.14 on_requested_incompatible_qos()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_requested_incompatible_qos (
             dds::sub::AnyDataReader & ,
            const  dds::core::status::RequestedIncompatibleQosStatus &  )  [inline], [virtual]
```

No-op.

Reimplemented from **dds::sub::NoOpSubscriberListener** (p. 1865).

8.216.2.15 on_sample_rejected()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_sample_rejected (
              dds::sub::AnyDataReader & ,
             const  dds::core::status::SampleRejectedStatus &  )  [inline], [virtual]
```

No-op.

Reimplemented from **dds::sub::NoOpSubscriberListener** (p. 1865).

8.216.2.16 on_liveliness_changed()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_liveliness_changed (
              dds::sub::AnyDataReader & ,
             const  dds::core::status::LivelinessChangedStatus &  )  [inline], [virtual]
```

No-op.

Reimplemented from **dds::sub::NoOpSubscriberListener** (p. 1866).

8.216.2.17 on_data_available()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_data_available (
              dds::sub::AnyDataReader &  )  [inline], [virtual]
```

No-op.

Reimplemented from **dds::sub::NoOpSubscriberListener** (p. 1866).

8.216.2.18 on_subscription_matched()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_subscription_matched (
              dds::sub::AnyDataReader & ,
             const  dds::core::status::SubscriptionMatchedStatus &  )  [inline], [virtual]
```

No-op.

Reimplemented from **dds::sub::NoOpSubscriberListener** (p. 1866).

8.216.2.19 on_sample_lost()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_sample_lost (
              dds::sub::AnyDataReader & ,
             const  dds::core::status::SampleLostStatus &  )  [inline], [virtual]
```

No-op.

Reimplemented from **dds::sub::NoOpSubscriberListener** (p. 1866).

8.216.2.20 on_inconsistent_topic()

```
virtual void dds::domain::NoOpDomainParticipantListener::on_inconsistent_topic (
              dds::topic::AnyTopic & ,
             const  dds::core::status::InconsistentTopicStatus &  )  [inline], [virtual]
```

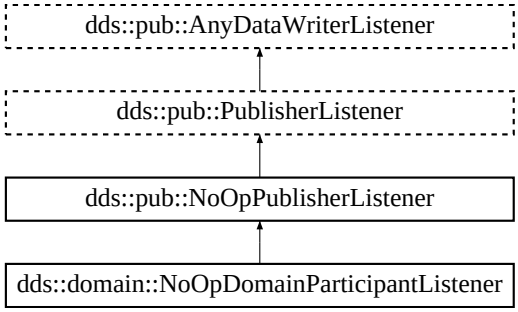
No-op.

8.217 dds::pub::NoOpPublisherListener Class Reference

A convenience implementation of **PublisherListener** (p. 2006) where all methods are overridden to do nothing.

```
#include <dds/pub/PublisherListener.hpp>
```

Inheritance diagram for dds::pub::NoOpPublisherListener:



Public Member Functions

- virtual void **on_offered_deadline_missed** (**dds::pub::AnyDataWriter** &, const **::dds::core::status::↩ OfferedDeadlineMissedStatus** &)

  No-op.
- virtual void **on_offered_incompatible_qos** (**dds::pub::AnyDataWriter** &, const **::dds::core::status::↩ OfferedIncompatibleQosStatus** &)

  No-op.
- virtual void **on_liveliness_lost** (**dds::pub::AnyDataWriter** &, const **::dds::core::status::LivelinessLost↩ Status** &)

  No-op.
- virtual void **on_publication_matched** (**dds::pub::AnyDataWriter** &, const **::dds::core::status::Publication↩ MatchedStatus** &)

  No-op.
- virtual void **on_reliable_writer_cache_changed** (**dds::pub::AnyDataWriter** &, const **rti::core::status::↩ ReliableWriterCacheChangedStatus** &)

  No-op.
- virtual void **on_reliable_reader_activity_changed** (**dds::pub::AnyDataWriter** &, const **rti::core::status::↩ ReliableReaderActivityChangedStatus** &)

  No-op.
- virtual void **on_sample_removed** (**dds::pub::AnyDataWriter** &, const **rti::core::Cookie** &)

  No-op.
- virtual void **on_instance_replaced** (**dds::pub::AnyDataWriter** &, const **dds::core::InstanceHandle** &)

  No-op.
- virtual void **on_application_acknowledgment** (**dds::pub::AnyDataWriter** &, const **rti::pub::↩ AcknowledgmentInfo** &)

  No-op.
- virtual void **on_service_request_accepted** (**dds::pub::AnyDataWriter** &, const **rti::core::status::Service↩ RequestAcceptedStatus** &)

  No-op.

8.217.1 Detailed Description

A convenience implementation of **PublisherListener** (p. 2006) where all methods are overridden to do nothing.

Most of the time, you can derive your listener from this class so you don't have to implement the methods you don't need.

8.217.2 Member Function Documentation

8.217.2.1 on_offered_deadline_missed()

```
virtual void dds::pub::NoOpPublisherListener::on_offered_deadline_missed (
             dds::pub::AnyDataWriter & ,
            const  ::dds::core::status::OfferedDeadlineMissedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::AnyDataWriterListener** (p. 822).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1847).

8.217.2.2 on_offered_incompatible_qos()

```
virtual void dds::pub::NoOpPublisherListener::on_offered_incompatible_qos (
             dds::pub::AnyDataWriter & ,
            const  ::dds::core::status::OfferedIncompatibleQosStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::AnyDataWriterListener** (p. 822).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1847).

8.217.2.3 on_liveliness_lost()

```
virtual void dds::pub::NoOpPublisherListener::on_liveliness_lost (
             dds::pub::AnyDataWriter & ,
            const  ::dds::core::status::LivelinessLostStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::AnyDataWriterListener** (p. 822).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1847).

8.217.2.4 on_publication_matched()

```
virtual void dds::pub::NoOpPublisherListener::on_publication_matched (
             dds::pub::AnyDataWriter & ,
            const  ::dds::core::status::PublicationMatchedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::AnyDataWriterListener** (p. 823).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1848).

8.217.2.5 on_reliable_writer_cache_changed()

```
virtual void dds::pub::NoOpPublisherListener::on_reliable_writer_cache_changed (
              dds::pub::AnyDataWriter & ,
             const  rti::core::status::ReliableWriterCacheChangedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::AnyDataWriterListener** (p. 823).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1848).

8.217.2.6 on_reliable_reader_activity_changed()

```
virtual void dds::pub::NoOpPublisherListener::on_reliable_reader_activity_changed (
              dds::pub::AnyDataWriter & ,
             const  rti::core::status::ReliableReaderActivityChangedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::AnyDataWriterListener** (p. 823).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1848).

8.217.2.7 on_sample_removed()

```
virtual void dds::pub::NoOpPublisherListener::on_sample_removed (
              dds::pub::AnyDataWriter & ,
             const  rti::core::Cookie &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::AnyDataWriterListener** (p. 823).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1848).

8.217.2.8 on_instance_replaced()

```
virtual void dds::pub::NoOpPublisherListener::on_instance_replaced (
              dds::pub::AnyDataWriter & ,
             const  dds::core::InstanceHandle &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::AnyDataWriterListener** (p. 824).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1849).

8.217.2.9 on_application_acknowledgment()

```
virtual void dds::pub::NoOpPublisherListener::on_application_acknowledgment (
             dds::pub::AnyDataWriter & ,
            const  rti::pub::AcknowledgmentInfo &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::AnyDataWriterListener** (p. 824).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1849).

8.217.2.10 on_service_request_accepted()

```
virtual void dds::pub::NoOpPublisherListener::on_service_request_accepted (
             dds::pub::AnyDataWriter & ,
            const  rti::core::status::ServiceRequestAcceptedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::pub::AnyDataWriterListener** (p. 824).

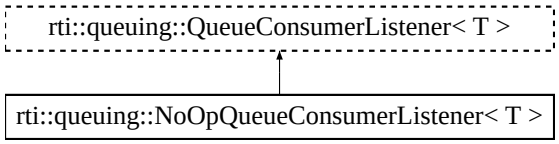
Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1849).

8.218 rti::queuing::NoOpQueueConsumerListener< T > Class Template Reference

A listener with an empty implementation of all methods.

`#include <QueueConsumerListener.hpp>`

Inheritance diagram for rti::queuing::NoOpQueueConsumerListener< T >:



Public Member Functions

- virtual void **on_sample_available** (**QueueConsumer**< T > &)

  User callback.
- virtual void **on_shared_reader_queue_matched** (**QueueConsumer**< T > &, const **dds::core::status::SubscriptionMatchedStatus** &)

  User callback.

8.218.1 Detailed Description

template<typename T>
class rti::queuing::NoOpQueueConsumerListener< T >

A listener with an empty implementation of all methods.

You can derive your listener from this class so you don't have to implement the methods you don't need.

8.218.2 Member Function Documentation

8.218.2.1 on_sample_available()

```
template<typename T >
virtual void  rti::queuing::NoOpQueueConsumerListener< T >::on_sample_available (
              QueueConsumer< T > &  )  [inline], [virtual]
```

User callback.

This callback is invoked whenever the **QueueConsumer** (p. 2063) has received at least one sample. Any operation to get samples i.e. QueueConsumer::take_sample) can be called within this context.

See also

dds::sub::DataReaderListener::on_data_available (p. 1058)

Implements **rti::queuing::QueueConsumerListener< T >** (p. 2076).

8.218.2.2 on_shared_reader_queue_matched()

```
template<typename T >
virtual void  rti::queuing::NoOpQueueConsumerListener< T >::on_shared_reader_queue_matched (
              QueueConsumer< T > & ,
             const  dds::core::status::SubscriptionMatchedStatus &  )  [inline], [virtual]
```

User callback.

This callback is invoked whenever a new SharedReaderQueue hosted by Queuing Service has matched the **Queue↩ Consumer** (p. 2063), or if an existing matching SharedReaderQueue is disposed.

See also

dds::core::status::SubscriptionMatchedStatus (p. 2439)

dds::sub::DataReaderListener::on_subscription_matched (p. 1058)

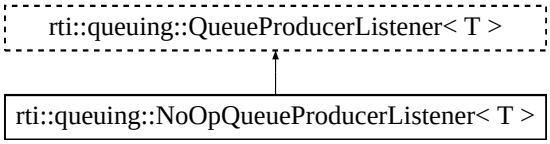
Implements **rti::queuing::QueueConsumerListener< T >** (p. 2077).

8.219 rti::queuing::NoOpQueueProducerListener< T > Class Template Reference

A listener with an empty implementation of all methods.

`#include <QueueProducerListener.hpp>`

Inheritance diagram for rti::queuing::NoOpQueueProducerListener< T >:



Public Member Functions

- virtual void **on_sample_acknowledged** (**QueueProducer**< T > &, const **rti::pub::AcknowledgmentInfo** &)
  User callback.
- virtual void **on_shared_reader_queue_matched** (**QueueProducer**< T > &, const **dds::core::status::PublicationMatchedStatus** &)
  User callback.

8.219.1 Detailed Description

template<**typename T**>
class rti::queuing::NoOpQueueProducerListener< **T** >

A listener with an empty implementation of all methods.

You can derive your listener from this class so you don't have to implement the methods you don't need.

8.219.2 Member Function Documentation

8.219.2.1 on_sample_acknowledged()

```
template<typename T >
virtual void  rti::queuing::NoOpQueueProducerListener< T >::on_sample_acknowledged (
              QueueProducer< T > & ,
             const  rti::pub::AcknowledgmentInfo &  )  [inline], [virtual]
```

User callback.

This callback is invoked whenever Queuing Service acknowledges a sample sent by the **QueueProducer** (p. 2080).

See also

AcknowledgmentInfo

Implements **rti::queuing::QueueProducerListener**< **T** > (p. 2091).

8.219.2.2 on_shared_reader_queue_matched()

```
template<typename T >
virtual void  rti::queuing::NoOpQueueProducerListener< T >::on_shared_reader_queue_matched (
              QueueProducer< T > & ,
             const  dds::core::status::PublicationMatchedStatus &  )  [inline], [virtual]
```

User callback.

This callback is invoked whenever a new SharedReaderQueue hosted by Queuing Service has matched the **Queue↩ Producer** (p. 2080), or if an existing matching SharedReaderQueue is disposed.

See also

dds::core::status::PublicationMatchedStatus (p. 1990)

dds::pub::DataWriterListener::on_publication_matched (p. 1205)

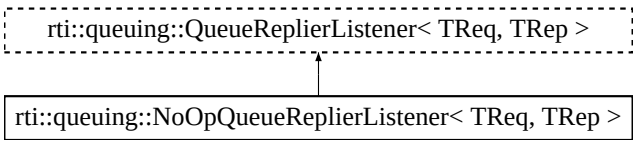
Implements **rti::queuing::QueueProducerListener< T >** (p. 2091).

8.220 rti::queuing::NoOpQueueReplierListener< TReq, TRep > Class Template Reference

A listener with an empty implementation of all methods.

```
#include <rti/queuing/QueueReplierListener.hpp>
```

Inheritance diagram for rti::queuing::NoOpQueueReplierListener< TReq, TRep >:



Public Member Functions

- virtual void **on_reply_acknowledged** (**QueueReplier**< TReq, TRep > &, const **rti::pub::Acknowledgment↩ Info** &)

  User callback.
- virtual void **on_request_available** (**QueueReplier**< TReq, TRep > &)

  User callback.
- virtual void **on_request_shared_reader_queue_matched** (**QueueReplier**< TReq, TRep > &, const **dds↩ ::core::status::SubscriptionMatchedStatus** &)

  User callback.
- virtual void **on_reply_shared_reader_queue_matched** (**QueueReplier**< TReq, TRep > &, const **dds::core↩ ::status::PublicationMatchedStatus** &)

  User callback.

8.220.1 Detailed Description

template<typename TReq, typename TRep>
class rti::queuing::NoOpQueueReplierListener< TReq, TRep >

A listener with an empty implementation of all methods.

You can derive your listener from this class so you don't have to implement the methods you don't need.

8.220.2 Member Function Documentation

8.220.2.1 on_reply_acknowledged()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::NoOpQueueReplierListener< TReq, TRep >::on_reply_acknowledged (
              QueueReplier< TReq, TRep > & producer,
             const  rti::pub::AcknowledgmentInfo & info )  [inline], [virtual]
```

User callback.

See also

QueueProducerListener::on_sample_acknowledged (p. 2091)

Implements **rti::queuing::QueueReplierListener< TReq, TRep >** (p. 2108).

8.220.2.2 on_request_available()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::NoOpQueueReplierListener< TReq, TRep >::on_request_available (
              QueueReplier< TReq, TRep > & requester )  [inline], [virtual]
```

User callback.

See also

QueueConsumerListener::on_sample_available (p. 2076)

Implements **rti::queuing::QueueReplierListener< TReq, TRep >** (p. 2108).

8.220.2.3 on_request_shared_reader_queue_matched()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::NoOpQueueReplierListener< TReq, TRep >::on_request_shared_reader_↩
queue_matched (
              QueueReplier< TReq, TRep > & requester,
             const  dds::core::status::SubscriptionMatchedStatus & status )  [inline], [virtual]
```

User callback.

See also

QueueConsumerListener::on_shared_reader_queue_matched (p. 2077)

Implements **rti::queuing::QueueReplierListener< TReq, TRep >** (p. 2108).

8.220.2.4 on_reply_shared_reader_queue_matched()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::NoOpQueueReplierListener< TReq, TRep >::on_reply_shared_reader_↩
queue_matched (
              QueueReplier< TReq, TRep > & requester,
             const  dds::core::status::PublicationMatchedStatus & status )  [inline], [virtual]
```

User callback.

See also

QueueProducerListener::on_shared_reader_queue_matched (p. 2091)

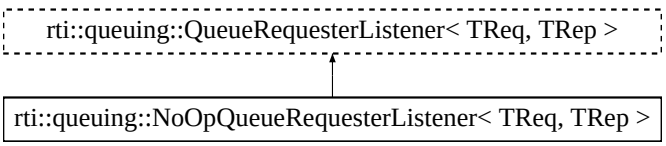
Implements **rti::queuing::QueueReplierListener< TReq, TRep >** (p. 2109).

8.221 rti::queuing::NoOpQueueRequesterListener< TReq, TRep > Class Template Reference

A listener with an empty implementation of all methods.

```
#include <rti/queuing/QueueRequesterListener.hpp>
```

Inheritance diagram for rti::queuing::NoOpQueueRequesterListener< TReq, TRep >:



Public Member Functions

- virtual void **on_request_acknowledged** (**QueueRequester**< TReq, TRep > &, const **rti::pub::↩ AcknowledgmentInfo** &)

  User callback.
- virtual void **on_reply_available** (**QueueRequester**< TReq, TRep > &)

  User callback.
- virtual void **on_request_shared_reader_queue_matched** (**QueueRequester**< TReq, TRep > &, const **dds↩ ::core::status::PublicationMatchedStatus** &)

  User callback.
- virtual void **on_reply_shared_reader_queue_matched** (**QueueRequester**< TReq, TRep > &, const **dds↩ ::core::status::SubscriptionMatchedStatus** &)

  User callback.

8.221.1 Detailed Description

template<typename TReq, typename TRep>
class rti::queuing::NoOpQueueRequesterListener< TReq, TRep >

A listener with an empty implementation of all methods.

You can derive your listener from this class so you don't have to implement the methods you don't need.

8.221.2 Member Function Documentation

8.221.2.1 on_request_acknowledged()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::NoOpQueueRequesterListener< TReq, TRep >::on_request_acknowledged (
              QueueRequester< TReq, TRep > & ,
             const  rti::pub::AcknowledgmentInfo &  )  [inline], [virtual]
```

User callback.

See also

QueueProducerListener::on_sample_acknowledged (p. 2091)

Implements **rti::queuing::QueueRequesterListener< TReq, TRep >** (p. 2127).

8.221.2.2 on_reply_available()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::NoOpQueueRequesterListener< TReq, TRep >::on_reply_available (
              QueueRequester< TReq, TRep > &  )  [inline], [virtual]
```

User callback.

See also

QueueConsumerListener::on_sample_available (p. 2076)

Implements **rti::queuing::QueueRequesterListener< TReq, TRep >** (p. 2127).

8.221.2.3 on_request_shared_reader_queue_matched()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::NoOpQueueRequesterListener< TReq, TRep >::on_request_shared_reader_↩
queue_matched (
              QueueRequester< TReq, TRep > & ,
             const  dds::core::status::PublicationMatchedStatus &  )  [inline], [virtual]
```

User callback.

See also

QueueProducerListener::on_shared_reader_queue_matched (p. 2091)

Implements **rti::queuing::QueueRequesterListener< TReq, TRep >** (p. 2127).

8.221.2.4 on_reply_shared_reader_queue_matched()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::NoOpQueueRequesterListener< TReq, TRep >::on_reply_shared_reader_↩
queue_matched (
              QueueRequester< TReq, TRep > & ,
             const  dds::core::status::SubscriptionMatchedStatus &  )  [inline], [virtual]
```

User callback.

See also

QueueConsumerListener::on_shared_reader_queue_matched (p. 2077)

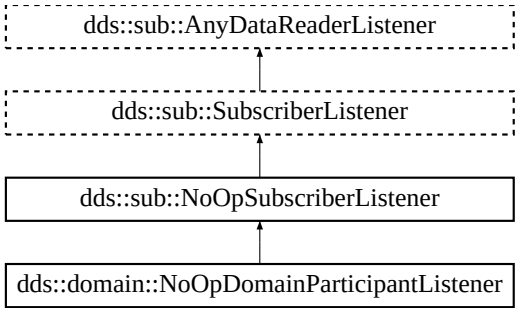
Implements **rti::queuing::QueueRequesterListener< TReq, TRep >** (p. 2128).

8.222 dds::sub::NoOpSubscriberListener Class Reference

A convenience implementation of **SubscriberListener** (p. 2422) where all methods are overridden to do nothing.

`#include <dds/sub/SubscriberListener.hpp>`

Inheritance diagram for dds::sub::NoOpSubscriberListener:



Public Member Functions

- virtual void **on_data_on_readers** (**Subscriber** &)

  No-op.
- virtual void **on_requested_deadline_missed** (**AnyDataReader** &, const **dds::core::status::Requested↩ DeadlineMissedStatus** &)

  No-op.
- virtual void **on_requested_incompatible_qos** (**AnyDataReader** &, const **dds::core::status::Requested↩ IncompatibleQosStatus** &)

  No-op.
- virtual void **on_sample_rejected** (**AnyDataReader** &, const **dds::core::status::SampleRejectedStatus** &)

  No-op.
- virtual void **on_liveliness_changed** (**AnyDataReader** &, const **dds::core::status::LivelinessChangedStatus** &)

  No-op.
- virtual void **on_data_available** (**AnyDataReader** &)

  No-op.
- virtual void **on_subscription_matched** (**AnyDataReader** &, const **dds::core::status::Subscription↩ MatchedStatus** &)

  No-op.
- virtual void **on_sample_lost** (**AnyDataReader** &, const **dds::core::status::SampleLostStatus** &)

  No-op.

8.222.1 Detailed Description

A convenience implementation of **SubscriberListener** (p. 2422) where all methods are overridden to do nothing.

Most of the time, you can derive your listener from this class so you don't have to implement the methods you don't need.

8.222.2 Member Function Documentation

8.222.2.1 on_data_on_readers()

```
virtual void dds::sub::NoOpSubscriberListener::on_data_on_readers (
             Subscriber &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::SubscriberListener** (p. 2422).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1850).

8.222.2.2 on_requested_deadline_missed()

```
virtual void dds::sub::NoOpSubscriberListener::on_requested_deadline_missed (
             AnyDataReader & ,
            const  dds::core::status::RequestedDeadlineMissedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::AnyDataReaderListener** (p. 813).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1850).

8.222.2.3 on_requested_incompatible_qos()

```
virtual void dds::sub::NoOpSubscriberListener::on_requested_incompatible_qos (
             AnyDataReader & ,
            const  dds::core::status::RequestedIncompatibleQosStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::AnyDataReaderListener** (p. 813).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1850).

8.222.2.4 on_sample_rejected()

```
virtual void dds::sub::NoOpSubscriberListener::on_sample_rejected (
             AnyDataReader & ,
            const  dds::core::status::SampleRejectedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::AnyDataReaderListener** (p. 813).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1850).

8.222.2.5 on_liveliness_changed()

```
virtual void dds::sub::NoOpSubscriberListener::on_liveliness_changed (
             AnyDataReader & ,
            const  dds::core::status::LivelinessChangedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::AnyDataReaderListener** (p. 814).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1851).

8.222.2.6 on_data_available()

```
virtual void dds::sub::NoOpSubscriberListener::on_data_available (
             AnyDataReader &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::AnyDataReaderListener** (p. 814).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1851).

8.222.2.7 on_subscription_matched()

```
virtual void dds::sub::NoOpSubscriberListener::on_subscription_matched (
             AnyDataReader & ,
            const  dds::core::status::SubscriptionMatchedStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::AnyDataReaderListener** (p. 814).

Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1851).

8.222.2.8 on_sample_lost()

```
virtual void dds::sub::NoOpSubscriberListener::on_sample_lost (
              AnyDataReader & ,
             const  dds::core::status::SampleLostStatus &  )  [inline], [virtual]
```

No-op.

Implements **dds::sub::AnyDataReaderListener** (p. 814).

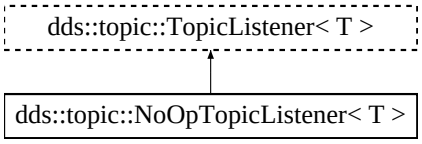
Reimplemented in **dds::domain::NoOpDomainParticipantListener** (p. 1851).

8.223 dds::topic::NoOpTopicListener< T > Class Template Reference

A convenience implementation of **TopicListener** (p. 2508) where all methods are overridden to do nothing.

```
#include <dds/topic/TopicListener.hpp>
```

Inheritance diagram for dds::topic::NoOpTopicListener< T >:



Public Member Functions

- virtual void **on_inconsistent_topic** (**Topic**< T > &, const **dds::core::status::InconsistentTopicStatus** &)
  No-op.

8.223.1 Detailed Description

template<**typename T**>
class dds::topic::NoOpTopicListener< T >

A convenience implementation of **TopicListener** (p. 2508) where all methods are overridden to do nothing.

8.223.2 Member Function Documentation

8.223.2.1 on_inconsistent_topic()

```
template<typename T >
virtual void  dds::topic::NoOpTopicListener< T >::on_inconsistent_topic (
              Topic< T > & ,
             const  dds::core::status::InconsistentTopicStatus &  )  [inline], [virtual]
```

No-op.

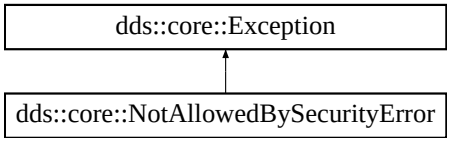
Implements **dds::topic::TopicListener< T >** (p. 2509).

8.224 dds::core::NotAllowedBySecurityError Class Reference

Indicates that an operation on the DDS API fails because the security plugins do not allow it.

`#include <Exception.hpp>`

Inheritance diagram for dds::core::NotAllowedBySecurityError:



Public Member Functions

- virtual const char * **what** () const throw ()

  *Access the message contained in this **NotAllowedBySecurityError** (p. 1868) exception.*

8.224.1 Detailed Description

Indicates that an operation on the DDS API fails because the security plugins do not allow it.

Inherits also from `std::logic_error`

An operation on the DDS API that fails because the security plugins do not allow it.

8.224.2 Member Function Documentation

8.224.2.1 what()

```
virtual const char * dds::core::NotAllowedBySecurityError::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **NotAllowedBySecurityError** (p. 1868) exception.

Returns

The message.

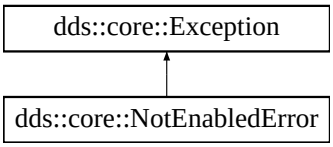
Implements **dds::core::Exception** (p. 1547).

8.225 dds::core::NotEnabledError Class Reference

A **NotEnabledError** (p. 1869) is thrown when an operation is invoked on a **dds::core::Entity** (p. 1520) that is not yet enabled.

```
#include <Exception.hpp>
```

Inheritance diagram for dds::core::NotEnabledError:



Public Member Functions

- virtual const char * **what** () const throw ()

  *Access the message contained in this **NotEnabledError** (p. 1869) exception.*

8.225.1 Detailed Description

A **NotEnabledError** (p. 1869) is thrown when an operation is invoked on a **dds::core::Entity** (p. 1520) that is not yet enabled.

Inherits also from `std::logic_error`

8.225.2 Member Function Documentation

8.225.2.1 what()

```
virtual const char * dds::core::NotEnabledError::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **NotEnabledError** (p. 1869) exception.

Returns

The message.

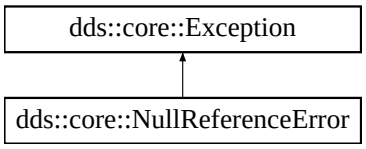
Implements **dds::core::Exception** (p. 1547).

8.226 dds::core::NullReferenceError Class Reference

Indicates an attempt to access a null object.

```
#include <Exception.hpp>
```

Inheritance diagram for dds::core::NullReferenceError:



Public Member Functions

- virtual const char * **what** () const throw ()
  Get a description of the error.

8.226.1 Detailed Description

Indicates an attempt to access a null object.

8.226.2 Member Function Documentation

8.226.2.1 what()

```
virtual const char * dds::core::NullReferenceError::what ( ) const throw ( )   [virtual]
```

Get a description of the error.

Implements **dds::core::Exception** (p. 1547).

8.227 rti::core::xtypes::octet_tag_t Struct Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* A special template type to obtain an IDL octet DynamicType from primitive_type

```
#include <Octet.hpp>
```

8.227.1 Detailed Description

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* A special template type to obtain an IDL octet DynamicType from primitive_type

Use this tag type as the template type for **dds::core::xtypes::primitive_type** (p. 634)<T> when creating a DynamicType for an IDL octet type. For example:

```
using namespace dds::core::xtypes;
StructType my_struct("MyStruct");
my_struct.add_member(Member("octet_member", primitive_type<rti::core::xtypes::octet_tag_t>()));
my_struct.add_member(Member("uint8_member", primitive_type<uint8_t>()));
```

8.228 dds::core::status::OfferedDeadlineMissedStatus Class Reference

Information about the status **dds::core::status::StatusMask::offered_deadline_missed()** (p. 2379)

```
#include <TStatus.hpp>
```

Public Member Functions

- int32_t **total_count** () const

  *Total cumulative count of the number of times the **dds::pub::DataWriter** (p. 1138) failed to write within its offered deadline.*
- int32_t **total_count_change** () const

  The incremental changes in total_count since the last time the listener was called or the status was read.
- const **dds::core::InstanceHandle last_instance_handle** () const

  *Handle to the last instance in the **dds::pub::DataWriter** (p. 1138) for which an offered deadline was missed.*

8.228.1 Detailed Description

Information about the status **dds::core::status::StatusMask::offered_deadline_missed()** (p. 2379)

Entity:

dds::pub::DataWriter (p. 1138)

Listener:

dds::pub::DataWriterListener (p. 1202)

The deadline that the **dds::pub::DataWriter** (p. 1138) has committed through its **dds::core::policy::Deadline** (p. 1255) was not respected for a specific instance.

8.228.2 Member Function Documentation

8.228.2.1 total_count()

```
int32_t dds::core::status::OfferedDeadlineMissedStatus::total_count ( ) const  [inline]
```

Total cumulative count of the number of times the **dds::pub::DataWriter** (p. 1138) failed to write within its offered deadline.

Missed deadlines accumulate; that is, each deadline period the `total_count` will be incremented by one.

8.228.2.2 total_count_change()

```
int32_t dds::core::status::OfferedDeadlineMissedStatus::total_count_change ( ) const  [inline]
```

The incremental changes in total_count since the last time the listener was called or the status was read.

8.228.2.3 last_instance_handle()

```
const  dds::core::InstanceHandle dds::core::status::OfferedDeadlineMissedStatus::last_instance_↩
handle ( ) const  [inline]
```

Handle to the last instance in the **dds::pub::DataWriter** (p. 1138) for which an offered deadline was missed.

8.229 dds::core::status::OfferedIncompatibleQosStatus Class Reference

Information about the status **dds::core::status::StatusMask::offered_incompatible_qos()** (p. 2380)

`#include <TStatus.hpp>`

Public Member Functions

- int32_t **total_count** () const

  *Total cumulative number of times the concerned **dds::pub::DataWriter** (p. 1138) discovered a **dds::sub::DataReader** (p. 976) for the same **dds::topic::Topic** (p. 2474), common partition with a requested QoS that is incompatible with that offered by the **dds::pub::DataWriter** (p. 1138).*
- int32_t **total_count_change** () const

  The incremental changes in total_count since the last time the listener was called or the status was read.
- **dds::core::policy::QosPolicyId last_policy_id** () const

  *The **dds::core::policy::QosPolicyId** (p. 514) of one of the policies that was found to be incompatible the last time an incompatibility was detected.*
- const **dds::core::policy::QosPolicyCountSeq policies** () const

  *A list containing for each policy the total number of times that the concerned **dds::pub::DataWriter** (p. 1138) discovered a **dds::sub::DataReader** (p. 976) for the same **dds::topic::Topic** (p. 2474) and common partition with a requested QoS that is incompatible with that offered by the **dds::pub::DataWriter** (p. 1138).*

8.229.1 Detailed Description

Information about the status **dds::core::status::StatusMask::offered_incompatible_qos()** (p. 2380)

Entity:

dds::pub::DataWriter (p. 1138)

Listener:

dds::pub::DataWriterListener (p. 1202)

The qos policy value was incompatible with what was requested.

8.229.2 Member Function Documentation

8.229.2.1 total_count()

```
int32_t dds::core::status::OfferedIncompatibleQosStatus::total_count ( ) const  [inline]
```

Total cumulative number of times the concerned **dds::pub::DataWriter** (p. 1138) discovered a **dds::sub::DataReader** (p. 976) for the same **dds::topic::Topic** (p. 2474), common partition with a requested QoS that is incompatible with that offered by the **dds::pub::DataWriter** (p. 1138).

8.229.2.2 total_count_change()

```
int32_t dds::core::status::OfferedIncompatibleQosStatus::total_count_change ( ) const  [inline]
```

The incremental changes in total_count since the last time the listener was called or the status was read.

8.229.2.3 last_policy_id()

```
dds::core::policy::QosPolicyId dds::core::status::OfferedIncompatibleQosStatus::last_policy_id (
) const  [inline]
```

The **dds::core::policy::QosPolicyId** (p. 514) of one of the policies that was found to be incompatible the last time an incompatibility was detected.

8.229.2.4 policies()

```
const  dds::core::policy::QosPolicyCountSeq dds::core::status::OfferedIncompatibleQosStatus↩
::policies ( ) const  [inline]
```

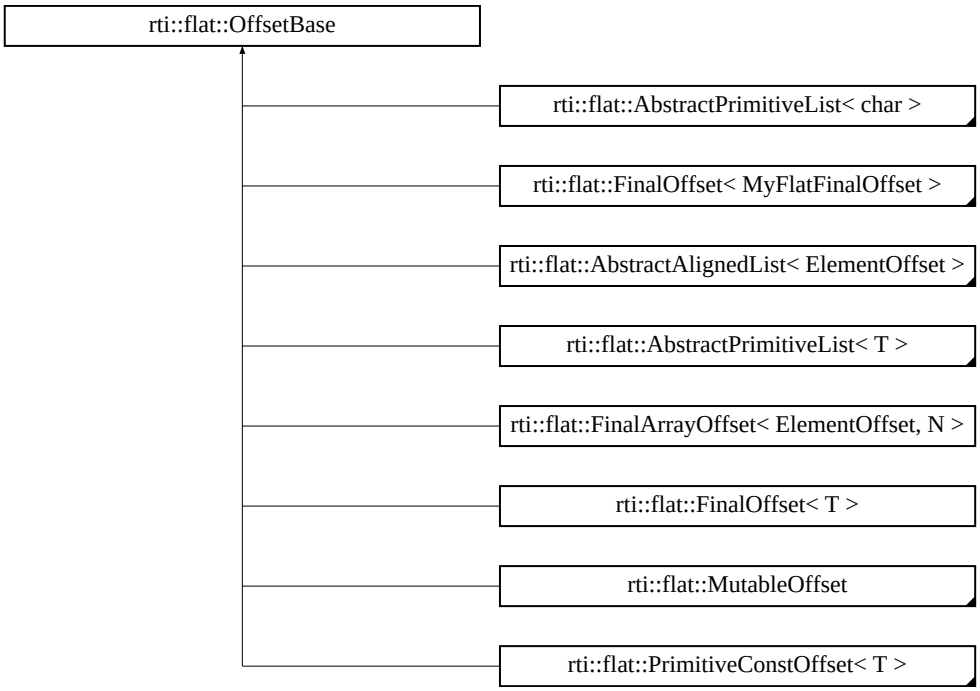
A list containing for each policy the total number of times that the concerned **dds::pub::DataWriter** (p. 1138) discovered a **dds::sub::DataReader** (p. 976) for the same **dds::topic::Topic** (p. 2474) and common partition with a requested QoS that is incompatible with that offered by the **dds::pub::DataWriter** (p. 1138).

8.230 rti::flat::OffsetBase Class Reference

Base class of all Offset types.

`#include <Offset.hpp>`

Inheritance diagram for rti::flat::OffsetBase:



Public Member Functions

- bool **is_null** () const

  Indicates whether this Offset doesn't point to a valid element.
- bool **is_cpp_compatible** () const

  *Indicates whether **rti::flat::plain_cast()** (p. 417) is possible.*
- const unsigned char * **get_buffer** () const

  Gets this member's position in the buffer.
- offset_t **get_buffer_size** () const

  Gets the size, in bytes, of this member in the buffer.

Friends

- bool **operator<** (const **OffsetBase** &s1, const **OffsetBase** &s2)

  Compares two Offsets.
- bool **operator>** (const **OffsetBase** &s1, const **OffsetBase** &s2)

  Compares two Offsets.

- bool **operator<=** (const **OffsetBase** &s1, const **OffsetBase** &s2)

  Compares two Offsets.
- bool **operator>=** (const **OffsetBase** &s1, const **OffsetBase** &s2)

  Compares two Offsets.
- bool **operator==** (const **OffsetBase** &s1, const **OffsetBase** &s2)

  Determines if two offets point to the same position.
- bool **operator!=** (const **OffsetBase** &s1, const **OffsetBase** &s2)

  Determines if two offets point to different positions.

8.230.1 Detailed Description

Base class of all Offset types.

See also

FlatData Offsets (p. 414)

8.230.2 Member Function Documentation

8.230.2.1 is_null()

```
bool rti::flat::OffsetBase::is_null ( ) const  [inline]
```

Indicates whether this Offset doesn't point to a valid element.

Comparing this object with `nullptr` is equivalent to calling **is_null()** (p. 1876). For example:

```
MyFlatMutableOffset my_mutable = ...;
auto my_final_member = my_mutable.my_final();
if (my_final_member.is_null()) { ...member doesn't exist... }
// alternative syntax:
if (my_final_member == nullptr) {...}
```

See also

Offset Error Management (p. 416)

Referenced by **get_buffer()**.

8.230.2.2 is_cpp_compatible()

```
bool rti::flat::OffsetBase::is_cpp_compatible ( ) const  [inline]
```

Indicates whether **rti::flat::plain_cast()** (p. 417) is possible.

Returns

True only if the data pointed to by this Offset can be **rti::flat::plain_cast()** (p. 417)

See also

rti::flat::plain_cast() (p. 417) for the requirements that a **type** (p. 2044) needs to meet

8.230.2.3 get_buffer()

```
const unsigned char * rti::flat::OffsetBase::get_buffer ( ) const  [inline]
```

Gets this member's position in the buffer.

Note

This function should be used for debugging purposes only. To access the data in this Offset use the Offset accessor methods or, if this type allows it, **rti::flat::plain_cast()** (p. 417).

Returns the position within the **Sample** (p. 2274)'s buffer that this Offset points to.

See also

get_buffer_size() (p. 1877)

References **is_null()**.

Referenced by **rti::flat::StringOffset::get_string()**.

8.230.2.4 get_buffer_size()

```
offset_t rti::flat::OffsetBase::get_buffer_size ( ) const  [inline]
```

Gets the size, in bytes, of this member in the buffer.

Returns the number of bytes that this member comprises after the position returned by get_buffer().

Referenced by **rti::flat::AbstractAlignedList< ElementOffset >::begin()**, and **rti::flat::AbstractAlignedList< ElementOffset >::end()**.

8.230.3 Friends And Related Function Documentation

8.230.3.1 operator<

```
bool operator< (
            const  OffsetBase & s1,
            const  OffsetBase & s2 )  [friend]
```

Compares two Offsets.

Returns

True if s1 points to a position smaller than s2's.

8.230.3.2 operator>

```
bool operator> (
            const  OffsetBase & s1,
            const  OffsetBase & s2 )  [friend]
```

Compares two Offsets.

Returns

True if s1 points to a position greater than s2's.

8.230.3.3 operator<=

```
bool operator<= (
            const  OffsetBase & s1,
            const  OffsetBase & s2 )  [friend]
```

Compares two Offsets.

Returns

True if s1 points to a position equal to or smaller than s2's.

8.230.3.4 operator>=

```
bool operator>= (
            const  OffsetBase & s1,
            const  OffsetBase & s2 )  [friend]
```

Compares two Offsets.

Returns

True if s1 points to a position equal to or greater than s2's.

8.230.3.5 operator==

```
bool operator== (
            const  OffsetBase & s1,
            const  OffsetBase & s2 )  [friend]
```

Determines if two offets point to the same position.

Returns

True if s1 points to the same position as s2.

8.230.3.6 operator"!=

```
bool operator!= (
            const  OffsetBase & s1,
            const  OffsetBase & s2 )  [friend]
```

Determines if two offets point to different positions.

Returns

True if s1 points to a different position than s2.

8.231 dds::core::optional< T > Class Template Reference

<<***value-type***>> *(*p. *350)* Represents an object that may not contain a valid value

```
#include <dds/core/Optional.hpp>
```

Public Member Functions

- **optional** () OMG_NOEXCEPT

  Create an unset optional object.
- **optional** (const T & **value**)

  Create an optional object with a copy of a value.
- **optional** (T && **value**)

  *<<**C++11**>> (p. 353) Create an optional object moving a value*
- **optional** (bool condition, const T & **value**)

  Create an optional member conditionally set or unset.
- **optional** (bool condition, T && **value**)

  *<<**C++11**>> (p. 353) Creates an optional member conditionally set or unset by moving a value.*
- ~**optional** ()

  Destroys the underlying object if it exists.
- void **set** (const T & **value**)

  Assigns a copy of an object.
- void **set** (T && **value**)

  *<<**C++11**>> (p. 353) Assigns an object by moving it*
- bool **is_set** () const OMG_NOEXCEPT

  Checks if this optional instance contains a valid object.
- bool **has_value** () const OMG_NOEXCEPT

  Checks if this optional instance contains a valid object.
- **operator bool** () const OMG_NOEXCEPT

  *<<**C++11**>> (p. 353) Returns **has_value()** (p. 1884)*
- void **reset** ()

  Destroys the underlying object and leaves this optional with an invalid (unset) value.
- const T & **value** () const

  Retrieves the underlying object if it exists.
- T & **value** ()

  Retrieves the underlying object if it exists.
- const T & **get** () const

  Retrieves the underlying object if it exists.
- T & **get** ()

  Retrieves the underlying object if it exists.
- const T * **get_ptr** () const

  *Returns &**get()** (p. 1886) if the object is initialized or NULL otherwise.*
- T * **get_ptr** ()

  *Returns &**get()** (p. 1886) if the object is initialized or NULL otherwise.*
- const T & **operator*** () const
- T & **operator*** ()
- const T * **operator->** () const
- T * **operator->** ()
- **optional**< T > & **operator=** (const **optional**< T > &other)

  Assignment operator.
- **optional**< T > & **operator=** (**optional**< T > &&other) OMG_NOEXCEPT

  *<<**C++11**>> (p. 353) Move-assignment operator*
- **optional**< T > & **operator=** (const T & **value**)

  Assign a (valid) value.
- **optional**< T > & **operator=** (T && **value**)

  Assign a (valid) value by moving an object.

Friends

- void **swap** (**optional**< T > &left, **optional**< T > &right) OMG_NOEXCEPT

  Swaps the underlying objects.

Related Functions

(Note that these are not member functions.)

- template<typename T >
  bool **operator==** (const **optional**< T > &a, const **optional**< T > &b)

  Compares two optional values.
- template<typename T >
  bool **operator!=** (const **optional**< T > &a, const **optional**< T > &b)

  Compares two optional values.
- template<typename T >
  bool **operator==** (const **optional**< T > &optional_value, const T & **value**)

  Compares an optional member and a value of the underlying type.
- template<typename T >
  bool **operator==** (const T & **value**, const **optional**< T > &optional_value)

  Compares an optional member and a value of the underlying type.
- template<typename T >
  bool **operator!=** (const **optional**< T > &optional_value, const T & **value**)

  Compares an optional member and a value of the underlying type.
- template<typename T >
  bool **operator!=** (const T & **value**, const **optional**< T > &optional_value)

  Compares an optional member and a value of the underlying type.
- template<typename T >
  std::ostream & **operator**<< (std::ostream &out, const **optional**< T > & **optional**)

  Applies `operator<<` *to* `*optional` *or to the string* `"NULL"` *if* `!optional`**`.has_value()`** (p. *1884*).

8.231.1 Detailed Description

template<typename T>
class dds::core::optional< T >

<<***value-type***>> (p. *350*) Represents an object that may not contain a valid value

Template Parameters

T	The type of the actual object this `optional<T>` wraps

Members of an **IDL** (p. 606) type marked with the `@optional` tag map to this C++ type.

When an optional value has a valid value **has_value()** (p. 1884) returns true and `operator*` returns a reference

to the actual object of type T. Otherwise **has_value()** (p. 1884) returns false and `operator*` throws **dds::core::↩ PreconditionNotMetError** (p. 1941). To assing a value you can use the assignment operator.

An optional object has full **value-type semantics** (p. 350); copying an optional value copies the underlying object if it exists.

This type's API is similar to that of `std::optional`.

See also

Working with IDL types (p. 606)

Examples

Foo.hpp.

8.231.2 Constructor & Destructor Documentation

8.231.2.1 optional() [1/5]

```
template<typename T >
dds::core::optional< T >::optional ( )  [inline]
```

Create an unset optional object.

Postcondition

!has_value()

8.231.2.2 optional() [2/5]

```
template<typename T >
dds::core::optional< T >::optional (
            const T & value )  [inline]
```

Create an optional object with a copy of a value.

Postcondition

has_value() (p. 1884) && *(*this) == value

8.231.2.3 optional() [3/5]

```
template<typename T >
dds::core::optional< T >::optional (
            T && value )  [inline]
```

<<**C++11**>> *(p. 353)* Create an optional object moving a value

Postcondition

has_value() (p. 1884)

8.231.2.4 optional() [4/5]

```
template<typename T >
dds::core::optional< T >::optional (
            bool condition,
            const T & value )  [inline]
```

Create an optional member conditionally set or unset.

Parameters

condition	If true creates an optional member with `value` otherwise it creates an unset optional member
value	The value to set if condition is true

Postcondition

has_value() (p. 1884) == condition

8.231.2.5 optional() [5/5]

```
template<typename T >
dds::core::optional< T >::optional (
            bool condition,
            T && value )  [inline]
```

<<**C++11**>> *(p. 353)* Creates an optional member conditionally set or unset by moving a value.

Moves the value rather than copying it.

See also

optional(bool, const T&) (p. 1883)

8.231.2.6 ~optional()

```
template<typename T >
dds::core::optional< T >::~ optional ( )  [inline]
```

Destroys the underlying object if it exists.

8.231.3 Member Function Documentation

8.231.3.1 set() [1/2]

```
template<typename T >
void  dds::core::optional< T >::set (
            const T & value )  [inline]
```

Assigns a copy of an object.

[DEPRECATED] Use `operator=` instead.

8.231.3.2 set() [2/2]

```
template<typename T >
void  dds::core::optional< T >::set (
            T && value )  [inline]
```

<<***C++11***>> *(p. 353)* Assigns an object by moving it

[DEPRECATED] Use `operator=` instead.

8.231.3.3 is_set()

```
template<typename T >
bool  dds::core::optional< T >::is_set ( ) const  [inline]
```

Checks if this optional instance contains a valid object.

[DEPRECATED] Use **`has_value()`** (p. 1884)

8.231.3.4 has_value()

```
template<typename T >
bool  dds::core::optional< T >::has_value ( ) const  [inline]
```

Checks if this optional instance contains a valid object.

If `my_optional.has_value()` is `true`, then `*my_optional` returns a valid object

Referenced by **dds::core::optional< T >::operator<<()**, and **dds::core::optional< T >::operator==()**.

8.231.3.5 operator bool()

```
template<typename T >
dds::core::optional< T >::operator bool ( ) const  [inline], [explicit]
```

<<***C++11***>> *(p. 353)* Returns **has_value()** (p. 1884)

8.231.3.6 reset()

```
template<typename T >
void  dds::core::optional< T >::reset ( )  [inline]
```

Destroys the underlying object and leaves this optional with an invalid (unset) value.

Postcondition

!has_value()

8.231.3.7 value() [1/2]

```
template<typename T >
const T &  dds::core::optional< T >::value ( ) const  [inline]
```

Retrieves the underlying object if it exists.

This operation, unlike `operator*` throws an exception if the underlying object doesn't exist.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !has_value().

8.231.3.8 value() [2/2]

```
template<typename T >
T &  dds::core::optional< T >::value ( )  [inline]
```

Retrieves the underlying object if it exists.

This operation, unlike `operator*` throws an exception if the underlying object doesn't exist.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !has_value().

8.231.3.9 get() [1/2]

```
template<typename T >
const T &  dds::core::optional< T >::get ( ) const  [inline]
```

Retrieves the underlying object if it exists.

[DEPRECATED] Use **value()** (p. 1886) or `operator*`

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !has_value().

Referenced by **dds::core::optional< T >::operator<<()**, and **dds::core::optional< T >::operator==()**.

8.231.3.10 get() [2/2]

```
template<typename T >
T &  dds::core::optional< T >::get ( )  [inline]
```

Retrieves the underlying object if it exists.

[DEPRECATED] Use **value()** (p. 1886) or `operator*`

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !has_value().

8.231.3.11 get_ptr() [1/2]

```
template<typename T >
const T *  dds::core::optional< T >::get_ptr ( ) const  [inline]
```

Returns &**get()** (p. 1886) if the object is initialized or NULL otherwise.

[DEPRECATED]

8.231.3.12 get_ptr() [2/2]

```
template<typename T >
T *  dds::core::optional< T >::get_ptr ( )  [inline]
```

Returns &**get()** (p. 1886) if the object is initialized or NULL otherwise.

[DEPRECATED]

8.231.3.13 operator*() [1/2]

```
template<typename T >
const T &  dds::core::optional< T >::operator* ( ) const  [inline]
```

Get the value, without checking if it exists

Precondition

has_value() (p. 1884), otherwise this operation has undefined behavior. See **value()** (p. 1886).

8.231.3.14 operator*() [2/2]

```
template<typename T >
T &  dds::core::optional< T >::operator* ( )  [inline]
```

Get the value, without checking if it exists

Precondition

has_value() (p. 1884), otherwise this operation has undefined behavior. See **value()** (p. 1886).

8.231.3.15 operator->() [1/2]

```
template<typename T >
const T *  dds::core::optional< T >::operator-> ( ) const  [inline]
```

Get the value.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !has_value()

8.231.3.16 operator->() [2/2]

```
template<typename T >
T *  dds::core::optional< T >::operator-> ( )  [inline]
```

Get the value.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !has_value()

8.231.3.17 operator=() [1/4]

```
template<typename T >
optional< T > &  dds::core::optional< T >::operator= (
             const  optional< T > & other )  [inline]
```

Assignment operator.

Copies *other if it exists.

8.231.3.18 operator=() [2/4]

```
template<typename T >
optional< T > &  dds::core::optional< T >::operator= (
              optional< T > && other )  [inline]
```

<<***C++11***>> *(p. 353)* Move-assignment operator

Moves *other if it exists.

8.231.3.19 operator=() [3/4]

```
template<typename T >
optional< T > &  dds::core::optional< T >::operator= (
             const T & value )  [inline]
```

Assign a (valid) value.

Parameters

value	The value to assign to this optional member

Postcondition

```
*(*this) == value
```

8.231.3.20 operator=() [4/4]

```
template<typename T >
optional< T > &  dds::core::optional< T >::operator= (
            T && value )  [inline]
```

Assign a (valid) value by moving an object.

Parameters

value	The value to move into this optional member

Postcondition

has_value() (p. 1884)

8.231.4 Friends And Related Function Documentation

8.231.4.1 swap

```
template<typename T >
void swap (
              optional< T > & left,
              optional< T > & right )  [friend]
```

Swaps the underlying objects.

This operation is always O(1).

8.231.4.2 operator==() [1/3]

```
template<typename T >
bool operator== (
            const  optional< T > & a,
            const  optional< T > & b )  [related]
```

Compares two optional values.

Returns

true if both are unset or both are set and `*a == *b`.

Examples

Foo.hpp.

References **dds::core::optional< T >::get()**, and **dds::core::optional< T >::has_value()**.

8.231.4.3 operator"!=() [1/3]

```
template<typename T >
bool operator!= (
            const  optional< T > & a,
            const  optional< T > & b )  [related]
```

Compares two optional values.

Returns

false if both are unset or both are set and `*a == *b`.

Examples

Foo.hpp.

8.231.4.4 operator==() [2/3]

```
template<typename T >
bool operator== (
            const  optional< T > & optional_value,
            const T & value )  [related]
```

Compares an optional member and a value of the underlying type.

Returns

Return true if `optional_value` is set and `*optional_value == value`

References **dds::core::optional< T >::get()**, and **dds::core::optional< T >::has_value()**.

8.231.4.5 operator==() [3/3]

```
template<typename T >
bool operator== (
            const T & value,
            const  optional< T > & optional_value )  [related]
```

Compares an optional member and a value of the underlying type.

Returns

Return true if optional_value is set and `*optional_value == value`

8.231.4.6 operator"!=() [2/3]

```
template<typename T >
bool operator!= (
            const  optional< T > & optional_value,
            const T & value )  [related]
```

Compares an optional member and a value of the underlying type.

Returns

Return false if optional_value is set and `*optional_value == value`

8.231.4.7 operator"!=() [3/3]

```
template<typename T >
bool operator!= (
            const T & value,
            const  optional< T > & optional_value )  [related]
```

Compares an optional member and a value of the underlying type.

Returns

Return false if optional_value is set and `*optional_value == value`

8.231.4.8 operator<<()

```
template<typename T >
std::ostream & operator<< (
            std::ostream & out,
            const  optional< T > & optional )  [related]
```

Applies operator<< to *optional or to the string "NULL" if !optional.**has_value()** (p. 1884).

References **dds::core::optional< T >::get()**, and **dds::core::optional< T >::has_value()**.

8.232 rti::core::optional_value< T > Class Template Reference

<<***extension***>> *(p. 354)* Represents a value that can be initialized or not

```
#include <OptionalValue.hpp>
```

Public Member Functions

- **optional_value** ()
  Creates an uninitialized value.
- **optional_value** (const T & **value**)
  Creates an instance with a value.
- **optional_value** (bool condition, const T & **value**)
  Conditionally creates an instance that can be uninitialized or initialized with a value.
- **optional_value** (const **optional_value**< T > &other)
  Copy constructor.
- bool **is_set** () const OMG_NOEXCEPT
  Returns true only if the value is initialized.
- bool **has_value** () const OMG_NOEXCEPT
  Returns true only if the value is initialized.
- **operator bool** () const OMG_NOEXCEPT
  <<***C++11***>> *(p. 353) Returns **has_value()** (p. 1894)*
- void **reset** ()
  *After calling this function, this **optional_value** (p. 1892) is not set.*
- const T & **value** () const
  Retrieves the underlying object if it exists.
- T & **value** ()
  Retrieves the underlying object if it exists.
- const T & **get** () const
  Retrieves the underlying object if it exists.
- T & **get** ()
  Retrieves the underlying object if it exists.
- const T & **operator*** () const
- T & **operator*** ()
- const T * **operator->** () const
- T * **operator->** ()

Friends

- void **swap** (**optional_value**< T > &left, **optional_value**< T > &right) OMG_NOEXCEPT
  Swaps the underlying objects.

8.232.1 Detailed Description

template<**typename T**>
class rti::core::optional_value< **T** >

<<***extension***>> *(p. 354)* Represents a value that can be initialized or not

This class is similar to **dds::core::optional** (p. 1879) and `std::optional`.

They have different implementations. **dds::core::optional** (p. 1879) is only used in IDL-generated types, while **rti::core::optional_value** (p. 1892) is used in parts of the API.

8.232.2 Constructor & Destructor Documentation

8.232.2.1 optional_value() [1/4]

```
template<typename T >
rti::core::optional_value< T >::optional_value ( )  [inline]
```

Creates an uninitialized value.

8.232.2.2 optional_value() [2/4]

```
template<typename T >
rti::core::optional_value< T >::optional_value (
            const T & value )  [inline]
```

Creates an instance with a value.

8.232.2.3 optional_value() [3/4]

```
template<typename T >
rti::core::optional_value< T >::optional_value (
            bool condition,
            const T & value )  [inline]
```

Conditionally creates an instance that can be uninitialized or initialized with a value.

Parameters

condition	If it is true it assigns `value`, otherwise this **optional_value** (p. 1892) is uninitialized
value	The value to use if `condition` is true.

References **rti::core::optional_value< T >::value()**.

8.232.2.4 optional_value() [4/4]

```
template<typename T >
rti::core::optional_value< T >::optional_value (
            const  optional_value< T > & other )  [inline]
```

Copy constructor.

This **optional_value** (p. 1892) will be initialized only if `other` is initialized.

References **rti::core::optional_value< T >::get()**, and **rti::core::optional_value< T >::has_value()**.

8.232.3 Member Function Documentation

8.232.3.1 is_set()

```
template<typename T >
bool  rti::core::optional_value< T >::is_set ( ) const  [inline]
```

Returns true only if the value is initialized.

[DEPRECATED] Use **has_value()** (p. 1894)

8.232.3.2 has_value()

```
template<typename T >
bool  rti::core::optional_value< T >::has_value ( ) const  [inline]
```

Returns true only if the value is initialized.

Referenced by **rti::core::optional_value< T >::optional_value()**, and **rti::core::optional_value< T >::value()**.

8.232.3.3 operator bool()

```
template<typename T >
rti::core::optional_value< T >::operator bool ( ) const  [inline], [explicit]
```

<<***C++11***>> *(p. 353)* Returns **has_value()** (p. 1894)

8.232.3.4 reset()

```
template<typename T >
void  rti::core::optional_value< T >::reset ( )  [inline]
```

After calling this function, this **optional_value** (p. 1892) is not set.

8.232.3.5 value() [1/2]

```
template<typename T >
const T &  rti::core::optional_value< T >::value ( ) const  [inline]
```

Retrieves the underlying object if it exists.

This operation, unlike `operator*` throws an exception if the underlying object doesn't exist.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !has_value().

References **rti::core::optional_value< T >::has_value()**.

Referenced by **rti::core::optional_value< T >::get()**, **rti::core::optional_value< T >::operator->()**, and **rti::core::optional_value< T >::optional_value()**.

8.232.3.6 value() [2/2]

```
template<typename T >
T &  rti::core::optional_value< T >::value ( )  [inline]
```

Retrieves the underlying object if it exists.

This operation, unlike `operator*` throws an exception if the underlying object doesn't exist.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !has_value().

References **rti::core::optional_value< T >::has_value()**.

8.232.3.7 get() [1/2]

```
template<typename T >
const T &  rti::core::optional_value< T >::get ( ) const  [inline]
```

Retrieves the underlying object if it exists.

[DEPRECATED] Use **value()** (p. 1895) or `operator*`

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !has_value().

References **rti::core::optional_value< T >::value()**.

Referenced by **rti::core::optional_value< T >::optional_value()**.

8.232.3.8 get() [2/2]

```
template<typename T >
T &  rti::core::optional_value< T >::get ( )  [inline]
```

Retrieves the underlying object if it exists.

[DEPRECATED] Use **value()** (p. 1895) or `operator*`

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !has_value().

References **rti::core::optional_value< T >::value()**.

8.232.3.9 operator*() [1/2]

```
template<typename T >
const T &  rti::core::optional_value< T >::operator* ( ) const  [inline]
```

Get the value, without checking if it exists

Precondition

has_value() (p. 1894), otherwise this operation has undefined behavior. See **value()** (p. 1895).

8.232.3.10 operator*() [2/2]

```
template<typename T >
T &  rti::core::optional_value< T >::operator* ( )  [inline]
```

Get the value, without checking if it exists

Precondition

has_value() (p. 1894), otherwise this operation has undefined behavior. See **value()** (p. 1895).

8.232.3.11 operator->() [1/2]

```
template<typename T >
const T *  rti::core::optional_value< T >::operator-> ( ) const  [inline]
```

Get the value.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !has_value()

References **rti::core::optional_value< T >::value()**.

8.232.3.12 operator->() [2/2]

```
template<typename T >
T *  rti::core::optional_value< T >::operator-> ( )  [inline]
```

Get the value.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if !has_value()

References **rti::core::optional_value< T >::value()**.

8.232.4 Friends And Related Function Documentation

8.232.4.1 swap

```
template<typename T >
void swap (
              optional_value< T > & left,
              optional_value< T > & right )  [friend]
```

Swaps the underlying objects.

It uses `swap(*left, *right)` if that function is defined for type `T`; otherwise it uses `std::swap`.

8.233 OsapiFile Class Reference

`#include <utilities.hpp>`

8.233.1 Detailed Description

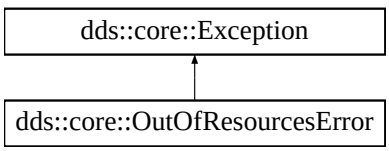
A class that wraps the RTIOsapiFile API to provide a C++-style file management. It automatically opens the file on construction, closes it on destruction, and removes the file when the object is destroyed.

8.234 dds::core::OutOfResourcesError Class Reference

Indicates that RTI Connext ran out of the resources needed to complete the operation.

`#include <Exception.hpp>`

Inheritance diagram for dds::core::OutOfResourcesError:



Public Member Functions

- virtual const char * **what** () const throw ()

  *Access the message contained in this **OutOfResourcesError** (p. 1898) exception.*

8.234.1 Detailed Description

Indicates that RTI Connext ran out of the resources needed to complete the operation.

Inherits also from `std::runtime_error`

8.234.2 Member Function Documentation

8.234.2.1 what()

```
virtual const char * dds::core::OutOfResourcesError::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **OutOfResourcesError** (p. 1898) exception.

Returns

The message.

Implements **dds::core::Exception** (p. 1547).

8.235 dds::core::policy::Ownership Class Reference

Specifies whether it is allowed for multiple **dds::pub::DataWriter** (p. 1138)'s to write the same instance of the data and if so, how these modifications should be arbitrated.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **Ownership** ()

  Creates an ownership policy set to shared.
- **Ownership** (**dds::core::policy::OwnershipKind** the_kind)

  Creates an instance with the specified ownership kind.
- **Ownership** & **kind** (**dds::core::policy::OwnershipKind** the_kind)

  Sets the ownership kind.
- **dds::core::policy::OwnershipKind kind** () const

  Getter (see setter with the same name)

Static Public Member Functions

- static **Ownership Exclusive** ()

  *Creates a **Ownership** (p. 1899) instance with exclusive kind.*
- static **Ownership Shared** ()

  *Creates a **Ownership** (p. 1899) instance with shared kind.*

8.235.1 Detailed Description

Specifies whether it is allowed for multiple **dds::pub::DataWriter** (p. 1138)'s to write the same instance of the data and if so, how these modifications should be arbitrated.

Entity:

dds::topic::Topic (p. 2474), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Status:

dds::core::status::StatusMask::offered_incompatible_qos() (p. 2380), **dds::core::status::StatusMask↩ ::requested_incompatible_qos()** (p. 2381)

Properties:

RxO (p. 514) = YES
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

See also

OWNERSHIP_STRENGTH (p. 539)

8.235.2 Usage

Along with the **OWNERSHIP_STRENGTH** (p. 539), this QoS policy specifies if **dds::sub::DataReader** (p. 976) entities can receive updates to the same instance (identified by its key) from multiple **dds::pub::DataWriter** (p. 1138) entities at the same time.

There are two kinds of ownership, selected by the setting of the `kind:` SHARED and EXCLUSIVE.

8.235.2.1 SHARED ownership

dds::core::policy::OwnershipKind_def::SHARED (p. 1906) indicates that RTI Connext does not enforce unique ownership for each instance. In this case, multiple writers can update the same data type instance. The subscriber to the **dds::topic::Topic** (p. 2474) will be able to access modifications from all **dds::pub::DataWriter** (p. 1138) objects, subject to the settings of other QoS that may filter particular samples (e.g. the **TIME_BASED_FILTER** (p. 547) or **HISTORY** (p. 533) policy). In any case, there is no "filtering" of modifications made based on the identity of the **dds::pub::Data↩ Writer** (p. 1138) that causes the modification.

8.235.2.2 EXCLUSIVE ownership

dds::core::policy::OwnershipKind_def::EXCLUSIVE (p. 1906) indicates that each instance of a data type can only be modified by one **dds::pub::DataWriter** (p. 1138). In other words, at any point in time, a single **dds::pub::DataWriter** (p. 1138) owns each instance and is the only one whose modifications will be visible to the **dds::sub::DataReader** (p. 976) objects. The owner is determined by selecting the **dds::pub::DataWriter** (p. 1138) with the highest value of the **dds::core::policy::OwnershipStrength::value** (p. 1908) that is currently alive, as defined by the **LIVELINESS** (p. 535) policy, and has not violated its **DEADLINE** (p. 523) contract with regards to the data instance.

Ownership (p. 1899) can therefore change as a result of:

- a **dds::pub::DataWriter** (p. 1138) in the system with a higher value of the strength that modifies the instance,
- a change in the strength value of the **dds::pub::DataWriter** (p. 1138) that owns the instance, and
- a change in the liveliness of the **dds::pub::DataWriter** (p. 1138) that owns the instance.
- a deadline with regards to the instance that is missed by the **dds::pub::DataWriter** (p. 1138) that owns the instance.

The behavior of the system is as if the determination was made independently by each **dds::sub::DataReader** (p. 976). Each **dds::sub::DataReader** (p. 976) may detect the change of ownership at a different time. It is not a requirement that at a particular point in time all the **dds::sub::DataReader** (p. 976) objects for that **dds::topic::Topic** (p. 2474) have a consistent picture of who owns each instance.

It is also not a requirement that the **dds::pub::DataWriter** (p. 1138) objects are aware of whether they own a particular instance. There is no error or notification given to a **dds::pub::DataWriter** (p. 1138) that modifies an instance it does not currently own.

The requirements are chosen to (a) preserve the decoupling of publishers and subscriber, and (b) allow the policy to be implemented efficiently.

It is possible that multiple **dds::pub::DataWriter** (p. 1138) objects with the same strength modify the same instance. If this occurs RTI Connext will pick one of the **dds::pub::DataWriter** (p. 1138) objects as the owner. It is not specified how the owner is selected. However, the algorithm used to select the owner guarantees that all **dds::sub::DataReader** (p. 976) objects will make the same choice of the particular **dds::pub::DataWriter** (p. 1138) that is the owner. It also guarantees that the owner remains the same until there is a change in strength, liveliness, the owner misses a deadline on the instance, or a new **dds::pub::DataWriter** (p. 1138) with higher same strength, or a new **dds::pub::DataWriter** (p. 1138) with same strength that should be deemed the owner according to the policy of the Service, modifies the instance.

Exclusive ownership is on an instance-by-instance basis. That is, a subscriber can receive values written by a lower strength **dds::pub::DataWriter** (p. 1138) as long as they affect instances whose values have not been set by the higher-strength **dds::pub::DataWriter** (p. 1138).

8.235.3 Compatibility

The value of the **dds::core::policy::OwnershipKind_def** (p. 1905) offered must exactly match the one requested or else they are considered incompatible.

8.235.4 Relationship between registration, liveliness and ownership

The need for registering/unregistering instances stems from two use cases:

- **Ownership** (p. 1899) resolution on redundant systems
- Detection of loss in topological connectivity

These two use cases also illustrate the semantic differences between the **dds::pub::DataWriter::unregister_instance** (p. 1160) and **dds::pub::DataWriter::dispose_instance()** (p. 1162).

8.235.4.1 Ownership Resolution on Redundant Systems

It is expected that users may use DDS to set up redundant systems where multiple **dds::pub::DataWriter** (p. 1138) entities are "capable" of writing the same instance. In this situation, the **dds::pub::DataWriter** (p. 1138) entities are configured such that:

- Either both are writing the instance "constantly"
- Or else they use some mechanism to classify each other as "primary" and "secondary", such that the primary is the only one writing, and the secondary monitors the primary and only writes when it detects that the primary "writer" is no longer writing.

Both cases above use the **dds::core::policy::OwnershipKind_def::EXCLUSIVE** (p. 1906) and arbitrate themselves by means of the **dds::core::policy::OwnershipStrength** (p. 1906). Regardless of the scheme, the desired behavior from the **dds::sub::DataReader** (p. 976) point of view is that **dds::sub::DataReader** (p. 976) normally receives data from the primary unless the "primary" writer stops writing, in which case the **dds::sub::DataReader** (p. 976) starts to receive data from the secondary **dds::pub::DataWriter** (p. 1138).

This approach requires some mechanism to detect that a **dds::pub::DataWriter** (p. 1138) (the primary) is no longer "writing" the data as it should. There are several reasons why this may happen and all must be detected (but not necessarily distinguished):

crash The writing process is no longer running (e.g. the whole application has crashed)

connectivity loss Connectivity to the writing application has been lost (e.g. network disconnection)

application fault The application logic that was writing the data is faulty and has stopped calling **dds::pub::DataWriter::write()** (p. 1148).

Arbitrating from a **dds::pub::DataWriter** (p. 1138) to one of a higher strength is simple and the decision can be taken autonomously by the **dds::sub::DataReader** (p. 976). Switching ownership from a higher strength **dds::pub::Data↩ Writer** (p. 1138) to one of a lower strength **dds::pub::DataWriter** (p. 1138) requires that the **dds::sub::DataReader** (p. 976) can make a determination that the stronger **dds::pub::DataWriter** (p. 1138) is "no longer writing the instance".

8.235.4.1.1 Case where the data is periodically updated This determination is reasonably simple when the data is being written periodically at some rate. The **dds::pub::DataWriter** (p. 1138) simply states its offered **dds::core::policy::Deadline** (p. 1255) (maximum interval between updates), and the **dds::sub::DataReader** (p. 976) automatically monitors that the **dds::pub::DataWriter** (p. 1138) indeed updates the instance at least once per **dds::core::policy::Deadline::period** (p. 1258) set by the **dds::sub::DataReader** (p. 976). If the deadline is missed, the **dds::sub::DataReader** (p. 976) considers the **dds::pub::DataWriter** (p. 1138) "not alive" and automatically gives ownership to the next highest-strength **dds::pub::DataWriter** (p. 1138) that *is* alive.

8.235.4.1.2 Case where data is not periodically updated The case where the **dds::pub::DataWriter** (p. 1138) is not writing data periodically is also a very important use-case. Since the instance is not being updated at any fixed period, the "deadline" mechanism cannot be used to determine ownership. The liveliness solves this situation. **Ownership** (p. 1899) is maintained while the **dds::pub::DataWriter** (p. 1138) is "alive" and for the **dds::pub::DataWriter** (p. 1138) to be alive it must fulfill its **dds::core::policy::Liveliness** (p. 1645) contract. The different means to renew liveliness (automatic, manual) combined by the implied renewal each time data is written handle the three conditions above [crash], [connectivity loss], and [application fault]. Note that to handle [application fault], LIVELINESS must be **dds::core::policy::LivelinessKind_def::MANUAL_BY_TOPIC** (p. 1655). The **dds::pub::DataWriter** (p. 1138) can retain ownership by periodically writing data or else calling assert_liveliness if it has no data to write. Alternatively if only protection against [crash] or [connectivity loss] is desired, it is sufficient that some task on the **dds::pub::DataWriter** (p. 1138) process periodically writes data or calls **dds::domain::DomainParticipant::assert_liveliness** (p. 1338). However, this scenario requires that the **dds::sub::DataReader** (p. 976) knows what instances are being "written" by the **dds::pub::DataWriter** (p. 1138). That is the only way that the **dds::sub::DataReader** (p. 976) deduces the ownership of specific instances from the fact that the **dds::pub::DataWriter** (p. 1138) is still "alive". Hence the need for the **dds::pub::DataWriter** (p. 1138) to "register" and "unregister" instances. Note that while "registration" can be done lazily the first time the **dds::pub::DataWriter** (p. 1138) writes the instance, "unregistration," in general, cannot. Similar reasoning will lead to the fact that unregistration will also require a message to be sent to the **dds::sub::DataReader** (p. 976).

8.235.4.2 Detection of Loss in Topological Connectivity

There are applications that are designed in such a way that their correct operation requires some minimal topological connectivity, that is, the writer needs to have a minimum number of readers or alternatively the reader must have a minimum number of writers.

A common scenario is that the application does not start doing its logic until it knows that some specific writers have the minimum configured readers (e.g the alarm monitor is up).

A *more* common scenario is that the application logic will wait until some writers appear that can provide some needed source of information (e.g. the raw sensor data that must be processed).

Furthermore, once the application is running it is a requirement that this minimal connectivity (from the source of the data) is monitored and the application informed if it is ever lost. For the case where data is being written periodically, the **dds::core::policy::Deadline** (p. 1255) and the on_deadline_missed listener provides the notification. The case where data is not periodically updated requires the use of the **dds::core::policy::Liveliness** (p. 1645) in combination with register_instance/unregister_instance to detect whether the "connectivity" has been lost, and the notification is provided by means of **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617).

In terms of the required mechanisms, the scenario is very similar to the case of maintaining ownership. In both cases, the reader needs to know whether a writer is still "managing the current value of an instance" even though it is not continually writing it and this knowledge requires the writer to keep its liveliness plus some means to know which instances the writer is currently "managing" (i.e. the registered instances).

8.235.4.3 Semantic Difference between unregister_instance and dispose

dds::pub::DataWriter::dispose_instance() (p. 1162) is semantically different from **dds::pub::DataWriter↩ ::unregister_instance** (p. 1160). **dds::pub::DataWriter::dispose_instance()** (p. 1162) indicates that the data instance no longer exists (e.g. a track that has disappeared, a simulation entity that has been destroyed, a record entry that has been deleted, etc.) whereas **dds::pub::DataWriter::unregister_instance** (p. 1160) indicates that the writer is no longer taking responsibility for updating the value of the instance.

Deleting a **dds::pub::DataWriter** (p. 1138) is equivalent to unregistering all the instances it was writing, but is *not* the same as "disposing" all the instances.

For a **dds::topic::Topic** (p. 2474) with **dds::core::policy::OwnershipKind_def::EXCLUSIVE** (p. 1906), if the current owner of an instance *disposes* it, the readers accessing the instance will see the instance_state as being "DISPOSED" and not see the values being written by the weaker writer (even after the stronger one has disposed the instance). This is because the **dds::pub::DataWriter** (p. 1138) that owns the instance is saying that the instance no longer exists (e.g. the master of the database is saying that a record has been deleted) and thus the readers should see it as such.

For a **dds::topic::Topic** (p. 2474) with **dds::core::policy::OwnershipKind_def::EXCLUSIVE** (p. 1906), if the current owner of an instance *unregisters* it, then it will relinquish ownership of the instance and thus the readers may see the value updated by another writer (which will then become the owner). This is because the owner said that it no longer will be providing values for the instance and thus another writer can take ownership and provide those values.

8.235.5 Constructor & Destructor Documentation

8.235.5.1 Ownership() [1/2]

```
dds::core::policy::Ownership::Ownership ( )  [inline]
```

Creates an ownership policy set to shared.

8.235.5.2 Ownership() [2/2]

```
dds::core::policy::Ownership::Ownership (
            dds::core::policy::OwnershipKind the_kind )  [inline], [explicit]
```

Creates an instance with the specified ownership kind.

8.235.6 Member Function Documentation

8.235.6.1 kind() [1/2]

```
Ownership & dds::core::policy::Ownership::kind (
            dds::core::policy::OwnershipKind the_kind )  [inline]
```

Sets the ownership kind.

[default] dds::core::policy::OwnershipKind_def::SHARED (p. 1906)

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Applied Per-Instance** (p. 287), **Data Availability** (p. 167)

8.235.6.2 kind() [2/2]

```
dds::core::policy::OwnershipKind dds::core::policy::Ownership::kind ( ) const  [inline]
```

Getter (see setter with the same name)

8.235.6.3 Exclusive()

```
static  Ownership dds::core::policy::Ownership::Exclusive ( )  [inline], [static]
```

Creates a **Ownership** (p. 1899) instance with exclusive kind.

8.235.6.4 Shared()

```
static  Ownership dds::core::policy::Ownership::Shared ( )  [inline], [static]
```

Creates a **Ownership** (p. 1899) instance with shared kind.

8.236 dds::core::policy::OwnershipKind_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) OwnershipKind.

```
#include <PolicyKind.hpp>
```

Public Types

- enum **type** {
  SHARED ,
  EXCLUSIVE }

  The underlying `enum` *type.*

8.236.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) OwnershipKind.

8.236.2 Member Enumeration Documentation

8.236.2.1 type

`enum` **dds::core::policy::OwnershipKind_def::type**

The underlying `enum` type.

Enumerator

SHARED	**[default]** Indicates shared ownership for each instance. Multiple writers are allowed to update the same instance and all the updates are made available to the readers. In other words there is no concept of an owner for the instances. This is the **default** behavior if the **OWNERSHIP** (p. 538) policy is not specified or supported.
EXCLUSIVE	Indicates each instance can only be owned by one **dds::pub::DataWriter** (p. 1138), but the owner of an instance can change dynamically. The selection of the owner is controlled by the setting of the **OWNERSHIP_STRENGTH** (p. 539) policy. The owner is always set to be the highest-strength **dds::pub::DataWriter** (p. 1138) object among the ones currently active (as determined by the **LIVELINESS** (p. 535)).

8.237 dds::core::policy::OwnershipStrength Class Reference

Specifies the value of the strength used to arbitrate among multiple **dds::pub::DataWriter** (p. 1138) objects that attempt to modify the same instance of a data type (identified by its **dds::topic::Topic** (p. 2474) and key).

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **OwnershipStrength** ()

  Creates an instance with the default strength (0)
- **OwnershipStrength** (int32_t strength)

  Creates an instance with the specified strength value.
- **OwnershipStrength** & **value** (int32_t strength)

  Sets the ownership strength to arbitrate among multiple writers.
- int32_t **value** () const

  Getter (see setter with the same name)

8.237.1 Detailed Description

Specifies the value of the strength used to arbitrate among multiple **dds::pub::DataWriter** (p. 1138) objects that attempt to modify the same instance of a data type (identified by its **dds::topic::Topic** (p. 2474) and key).

This policy only applies if the **OWNERSHIP** (p. 538) policy is of kind **dds::core::policy::OwnershipKind_def::↩ EXCLUSIVE** (p. 1906).

Entity:

dds::pub::DataWriter (p. 1138)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **YES** (p. 514)

The value of the **OWNERSHIP_STRENGTH** (p. 539) is used to determine the ownership of a data instance (identified by the key). The arbitration is performed by the **dds::sub::DataReader** (p. 976).

See also

EXCLUSIVE ownership (p. 1901)

8.237.2 Constructor & Destructor Documentation

8.237.2.1 OwnershipStrength() [1/2]

```
dds::core::policy::OwnershipStrength::OwnershipStrength ( )  [inline]
```

Creates an instance with the default strength (0)

8.237.2.2 OwnershipStrength() [2/2]

```
dds::core::policy::OwnershipStrength::OwnershipStrength (
            int32_t strength )  [inline], [explicit]
```

Creates an instance with the specified strength value.

8.237.3 Member Function Documentation

8.237.3.1 value() [1/2]

```
OwnershipStrength & dds::core::policy::OwnershipStrength::value (
            int32_t strength )  [inline]
```

Sets the ownership strength to arbitrate among multiple writers.

[default] 0

[range] [0, 1 million]

Categories

Mutable (p. 280), **Discovery Metadata** (p. 288), **Applied Per-Instance** (p. 287), **Data Availability** (p. 167)

8.237.3.2 value() [2/2]

```
int32_t dds::core::policy::OwnershipStrength::value ( ) const  [inline]
```

Getter (see setter with the same name)

8.238 dds::topic::ParticipantBuiltinTopicData Class Reference

Entry created when a **dds::domain::DomainParticipant** (p. 1326) is discovered.

```
#include <dds/topic/BuiltinTopic.hpp>
```

Public Member Functions

- **TParticipantBuiltinTopicData** ()

  *Create a default **ParticipantBuiltinTopicData** (p. 1908).*
- const **dds::topic::BuiltinTopicKey** & **key** () const

  Get the DCPS key to distinguish entries.
- const **dds::core::policy::UserData** & **user_data** () const

  *Get the **dds::core::policy::UserData** (p. 2591) policy of the corresponding **dds::domain::DomainParticipant** (p. 1326).*
- const **rti::core::policy::Property** & **property** () const

  *<<**extension**>> (p. 354) Get the name-value pair properties to be stored with dds::domin::DomainParticipant*
- const **rti::core::ProtocolVersion** & **rtps_protocol_version** () const

  *<<**extension**>> (p. 354) Get the version number of the RTPS wire protocol used.*
- const **rti::core::VendorId** & **rtps_vendor_id** () const

  *<<**extension**>> (p. 354) Get the ID of the vendor implementing the RTPS wire protocol.*
- uint32_t **dds_builtin_endpoints** () const

  *<<**extension**>> (p. 354) Bitmap of builtin endpoints supported by the participant*
- uint32_t **available_builtin_endpoints_ext** () const

  *<<**extension**>> (p. 354) Extended bitmap of builtin endpoints supported by the participant*
- const **dds::core::vector**< **rti::core::Locator** > & **default_unicast_locators** () const

  *<<**extension**>> (p. 354) Get the unicast locators used when individual entities do not specify unicast locators.*
- const **dds::core::policy::Partition** & **partition** () const

  *<<**extension**>> (p. 354) Get the Partition Qos Policy*
- const **rti::topic::trust::ParticipantTrustProtectionInfo** & **trust_protection_info** () const

  *<<**extension**>> (p. 354) Trust plugins information associated with the discovered DomainParticipant.*
- const **rti::topic::trust::ParticipantTrustAlgorithmInfo** & **trust_algorithm_info** () const

  *<<**extension**>> (p. 354) Trust plugins algorithms associated with the discovered DomainParticipant.*
- const **rti::core::ProductVersion** & **product_version** () const

  *<<**extension**>> (p. 354) Get the current version for RTI Connext.*
- const **rti::core::policy::EntityName** & **participant_name** () const

  *<<**extension**>> (p. 354) Get the participant name and role name.*
- int32_t **domain_id** () const

  *<<**extension**>> (p. 354) Get the domain ID associated with the discovered **dds::domain::DomainParticipant** (p. 1326).*
- const **rti::core::TransportInfoSeq** & **transport_info** () const

  *<<**extension**>> (p. 354) Get the sequence of **rti::core::TransportInfo** (p. 2541) containing information about each of the installed transports of the discovered **dds::domain::DomainParticipant** (p. 1326).*
- DDS_Boolean **partial_configuration** () const

  *<<**extension**>> (p. 354) Get the value of partial_configuration in the **ParticipantBuiltinTopicData** (p. 1908). This will be TRUE if the **ParticipantBuiltinTopicData** (p. 1908) only contains bootstrapping information.*
- const **dds::core::Duration** & **reachability_lease_duration** () const

  *<<**extension**>> (p. 354) Get the reachability lease duration.*
- uint32_t **vendor_builtin_endpoints** () const

  *<<**extension**>> (p. 354) Bitmap of vendor-specific builtin endpoints supported by the participant*

8.238.1 Detailed Description

Entry created when a **dds::domain::DomainParticipant** (p. 1326) is discovered.

8.238.2 Member Function Documentation

8.238.2.1 TParticipantBuiltinTopicData()

```
dds::topic::ParticipantBuiltinTopicData::TParticipantBuiltinTopicData ( )  [inline]
```

Create a default **ParticipantBuiltinTopicData** (p. 1908).

8.238.2.2 key()

```
const  dds::topic::BuiltinTopicKey & dds::topic::ParticipantBuiltinTopicData::key ( ) const  [inline]
```

Get the DCPS key to distinguish entries.

Returns

The key

8.238.2.3 user_data()

```
const  dds::core::policy::UserData & dds::topic::ParticipantBuiltinTopicData::user_data ( ) const
[inline]
```

Get the **dds::core::policy::UserData** (p. 2591) policy of the corresponding **dds::domain::DomainParticipant** (p. 1326).

Returns

The UserData policy

8.238.2.4 property()

```
const  rti::core::policy::Property & property ( ) const
```

<<***extension***>> *(p. 354)* Get the name-value pair properties to be stored with dds::domin::DomainParticipant

Returns

The name-value pair properties that were propagated from the corresponding **dds::domain::DomainParticipant** (p. 1326)

8.238.2.5 rtps_protocol_version()

```
const  rti::core::ProtocolVersion & rtps_protocol_version ( ) const
```

<<***extension***>> *(p. 354)* Get the version number of the RTPS wire protocol used.

Returns

The protocol version.

8.238.2.6 rtps_vendor_id()

```
const  rti::core::VendorId & rtps_vendor_id ( ) const
```

<<***extension***>> *(p. 354)* Get the ID of the vendor implementing the RTPS wire protocol.

Returns

The vendor id.

8.238.2.7 dds_builtin_endpoints()

```
uint32_t dds_builtin_endpoints ( ) const
```

<<***extension***>> *(p. 354)* Bitmap of builtin endpoints supported by the participant

Each bit indicates a builtin endpoint that may be available on the participant.

Bit	Endpoint
0	SPDPbuiltinParticipantWriter
1	SPDPbuiltinParticipantReader
2	SEDPbuiltinPublicationsWriter
3	SEDPbuiltinPublicationsReader
4	SEDPbuiltinSubscriptionsWriter
5	SEDPbuiltinSubscriptionsReader
10	BuiltinParticipantMessageWriter
11	BuiltinParticipantMessageReader
12	TypeLookupServiceRequestDataWriter
13	TypeLookupServiceRequestDataReader
14	TypeLookupServiceReplyDataWriter
15	TypeLookupServiceReplyDataReader
16	SEDPbuiltinPublicationsSecureWriter
17	SEDPbuiltinPublicationsSecureReader

Bit	Endpoint
18	SEDPbuiltinSubscriptionsSecureWriter
19	SEDPbuiltinSubscriptionsSecureReader
20	BuiltinParticipantMessageSecureWriter
21	BuiltinParticipantMessageSecureReader
22	BuiltinParticipantStatelessMessageWriter
23	BuiltinParticipantStatelessMessageReader
24	BuiltinParticipantVolatileMessageSecureWriter
25	BuiltinParticipantVolatileMessageSecureReader
26	SPDPbuiltinParticipantSecureWriter
27	SPDPbuiltinParticipantSecureReader

8.238.2.8 available_builtin_endpoints_ext()

```
uint32_t available_builtin_endpoints_ext ( ) const
```

<<***extension***>> *(p. 354)* Extended bitmap of builtin endpoints supported by the participant

This bitmap is an extension of the **dds::topic::ParticipantBuiltinTopicData::dds_builtin_endpoints** (p. 1911) field. It is used to indicate additional builtin endpoints that may be available on the participant.

Bit	Endpoint
0	TypeLookupServiceRequestSecureWriter
1	TypeLookupServiceRequestSecureReader
2	TypeLookupServiceReplySecureWriter
3	TypeLookupServiceReplySecureReader

8.238.2.9 default_unicast_locators()

```
const  dds::core::vector<  rti::core::Locator > & default_unicast_locators ( ) const
```

<<***extension***>> *(p. 354)* Get the unicast locators used when individual entities do not specify unicast locators.

Returns

A sequence of the default unicast locators used

8.238.2.10 partition()

```
const  dds::core::policy::Partition & partition ( ) const
```

<<***extension***>> *(*p. *354)* Get the Partition Qos Policy

Returns

The Partition QoS policy

8.238.2.11 trust_protection_info()

```
const  rti::topic::trust::ParticipantTrustProtectionInfo & trust_protection_info ( ) const
```

<<***extension***>> *(*p. *354)* Trust plugins information associated with the discovered DomainParticipant.

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged DomainParticipant data and metadata.

trust_protection_info contains information about how RTPS wire serialization, discovery, and liveliness interact with the loaded Trust Plugins. Two DomainParticipants will not match if their trust_protection_info is incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the DomainParticipant is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.238.2.12 trust_algorithm_info()

```
const  rti::topic::trust::ParticipantTrustAlgorithmInfo & trust_algorithm_info ( ) const
```

<<***extension***>> *(*p. *354)* Trust plugins algorithms associated with the discovered DomainParticipant.

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged DomainParticipant data and metadata. trust_algorithm_info contains information about what algorithms the loaded Trust Plugins are running. Two DomainParticipants will not match if their trust_algorithm_info are incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the DomainParticipant is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.238.2.13 product_version()

```
const  rti::core::ProductVersion & product_version ( ) const
```

<<***extension***>> *(*p. *354)* Get the current version for RTI Connext.

Returns

The RTI Connext version being used

8.238.2.14 participant_name()

```
const  rti::core::policy::EntityName & participant_name ( ) const
```

<<***extension***>> *(*p. *354)* Get the participant name and role name.

Returns

The **rti::core::policy::EntityName** (p. 1530) QoS policy for the corresponding **dds::domain::DomainParticipant** (p. 1326).

8.238.2.15 domain_id()

```
int32_t domain_id ( ) const
```

<<***extension***>> *(*p. *354)* Get the domain ID associated with the discovered **dds::domain::DomainParticipant** (p. 1326).

Returns

The domain id

Examples

Foo_publisher.cxx, and **Foo_subscriber.cxx**.

8.238.2.16 transport_info()

```
const  rti::core::TransportInfoSeq & transport_info ( ) const
```

<<***extension***>> *(p. 354)* Get the sequence of **rti::core::TransportInfo** (p. 2541) containing information about each of the installed transports of the discovered **dds::domain::DomainParticipant** (p. 1326).

This parameter contains a sequence of **rti::core::TransportInfo** (p. 2541) containing the class_id and message_↩ size_max for all installed transports of the discovered participant. The maximum number of **rti::core::TransportInfo** (p. 2541) that will be stored in this sequence is controlled by the DomainParticipant's resource limit **rti::core::policy::**↩ **DomainParticipantResourceLimits::transport_info_list_max_length** (p. 1428).

Returns

The sequence of transport infos

8.238.2.17 partial_configuration()

```
DDS_Boolean partial_configuration ( ) const
```

<<***extension***>> *(p. 354)* Get the value of partial_configuration in the **ParticipantBuiltinTopicData** (p. 1908). This will be TRUE if the **ParticipantBuiltinTopicData** (p. 1908) only contains bootstrapping information.

Returns

The value of partial_configuration

8.238.2.18 reachability_lease_duration()

```
const  dds::core::Duration & reachability_lease_duration ( ) const
```

<<***extension***>> *(p. 354)* Get the reachability lease duration.

8.238.2.19 vendor_builtin_endpoints()

```
uint32_t vendor_builtin_endpoints ( ) const
```

<<***extension***>> *(p. 354)* Bitmap of vendor-specific builtin endpoints supported by the participant

8.239 rti::test::ParticipantFactoryGuard Class Reference

RAII guard for managing the DomainParticipantFactory lifecycle.

`#include <utilities.hpp>`

8.239.1 Detailed Description

RAII guard for managing the DomainParticipantFactory lifecycle.

When an instance of this class goes out of scope, it automatically finalizes the participant factory, ensuring all DDS resources are properly released.

8.240 rti::topic::trust::ParticipantTrustAlgorithmInfo Class Reference

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins algorithm information associated with the discovered DomainParticipant.

`#include <rti/topic/trust/ParticipantTrustAlgorithmInfo.hpp>`

Public Member Functions

- **ParticipantTrustAlgorithmInfo** ()=default

  Create an instance with the default Trust Algorithm Info associated with the discovered DomainParticipant.
- **ParticipantTrustSignatureAlgorithmInfo signature** () const

  Get the Trust Plugins algorithms for validation of data and metadata exchanged by the DomainParticipant.
- **ParticipantTrustKeyEstablishmentAlgorithmInfo key_establishment** () const

  Get the Trust Plugins algorithms for transformation of data and metadata exchanged by the DomainParticipant.
- **ParticipantTrustInterceptorAlgorithmInfo interceptor** () const

  Get the Trust Plugins algorithms for interception of data and metadata exchanged by the DomainParticipant.

8.240.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins algorithm information associated with the discovered DomainParticipant.

8.240.2 Constructor & Destructor Documentation

8.240.2.1 ParticipantTrustAlgorithmInfo()

```
rti::topic::trust::ParticipantTrustAlgorithmInfo::ParticipantTrustAlgorithmInfo ( )  [default]
```

Create an instance with the default Trust Algorithm Info associated with the discovered DomainParticipant.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

8.240.3 Member Function Documentation

8.240.3.1 signature()

ParticipantTrustSignatureAlgorithmInfo rti::topic::trust::ParticipantTrustAlgorithmInfo::signature ( ) const [inline]

Get the Trust Plugins algorithms for validation of data and metadata exchanged by the DomainParticipant.

8.240.3.2 key_establishment()

ParticipantTrustKeyEstablishmentAlgorithmInfo rti::topic::trust::ParticipantTrustAlgorithmInfo↩ ::key_establishment ( ) const [inline]

Get the Trust Plugins algorithms for transformation of data and metadata exchanged by the DomainParticipant.

8.240.3.3 interceptor()

ParticipantTrustInterceptorAlgorithmInfo rti::topic::trust::ParticipantTrustAlgorithmInfo::interceptor ( ) const [inline]

Get the Trust Plugins algorithms for interception of data and metadata exchanged by the DomainParticipant.

8.241 rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo Class Reference

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Trust Plugins interception algorithm information associated with the discovered DomainParticipant.

```
#include <rti/topic/trust/ParticipantTrustInterceptorAlgorithmInfo.hpp>
```

Public Member Functions

- **ParticipantTrustInterceptorAlgorithmInfo** ()=default

  Create an instance with the default Trust Interceptor Algorithm Info associated with the discovered DomainParticipant.
- uint32_t **supported_mask** () const

  Get the Trust Plugins algorithms supported for interception of data and metadata exchanged by the DomainParticipant.
- uint32_t **builtin_endpoints_required_mask** () const

  Get the Trust Plugins algorithms used for interception of metadata exchanged by the builtin endpoints.
- uint32_t **builtin_kx_endpoints_required_mask** () const

  Get the Trust Plugins algorithms used for interception of metadata exchanged by the key exchange builtin endpoints.
- uint32_t **user_endpoints_default_required_mask** () const

  *Get the Trust Plugins algorithms used by default for interception of data exchanged by the user endpoints. This is the algorithm that remote DomainParticipants should assume when **rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo** (p. 1517) is not serialized.*

8.241.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins interception algorithm information associated with the discovered DomainParticipant.

8.241.2 Constructor & Destructor Documentation

8.241.2.1 ParticipantTrustInterceptorAlgorithmInfo()

```
rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo::ParticipantTrustInterceptorAlgorithm↩
Info ( )  [default]
```

Create an instance with the default Trust Interceptor Algorithm Info associated with the discovered DomainParticipant.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

8.241.3 Member Function Documentation

8.241.3.1 supported_mask()

```
uint32_t rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo::supported_mask ( ) const
[inline]
```

Get the Trust Plugins algorithms supported for interception of data and metadata exchanged by the DomainParticipant.

8.241.3.2 builtin_endpoints_required_mask()

```
uint32_t rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo::builtin_endpoints_required↩
_mask ( ) const  [inline]
```

Get the Trust Plugins algorithms used for interception of metadata exchanged by the builtin endpoints.

8.241.3.3 builtin_kx_endpoints_required_mask()

```
uint32_t rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo::builtin_kx_endpoints_↩
required_mask ( ) const  [inline]
```

Get the Trust Plugins algorithms used for interception of metadata exchanged by the key exchange builtin endpoints.

8.241.3.4 user_endpoints_default_required_mask()

```
uint32_t rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo::user_endpoints_default_↩
required_mask ( ) const  [inline]
```

Get the Trust Plugins algorithms used by default for interception of data exchanged by the user endpoints. This is the algorithm that remote DomainParticipants should assume when **rti::topic::trust::EndpointTrustInterceptorAlgorithm↩ Info** (p. 1517) is not serialized.

8.242 rti::topic::trust::ParticipantTrustKeyEstablishmentAlgorithmInfo Class Reference

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.

```
#include <rti/topic/trust/ParticipantTrustKeyEstablishmentAlgorithmInfo.hpp>
```

Public Member Functions

- **ParticipantTrustKeyEstablishmentAlgorithmInfo** ()=default

  Create an instance with the default Trust Key Establishment Algorithm Info associated with the discovered Domain↩ Participant.
- **TrustAlgorithmRequirements shared_secret** () const

  Get the Trust Plugins key establishment algorithm requirements of the DomainParticipant in the context of deriving a shared secret.

8.242.1 Detailed Description

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.

8.242.2 Constructor & Destructor Documentation

8.242.2.1 ParticipantTrustKeyEstablishmentAlgorithmInfo()

```
rti::topic::trust::ParticipantTrustKeyEstablishmentAlgorithmInfo::ParticipantTrustKeyEstablishment↩
AlgorithmInfo ( )  [default]
```

Create an instance with the default Trust Key Establishment Algorithm Info associated with the discovered Domain↩ Participant.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

8.242.3 Member Function Documentation

8.242.3.1 shared_secret()

```
TrustAlgorithmRequirements rti::topic::trust::ParticipantTrustKeyEstablishmentAlgorithmInfo↩
::shared_secret ( ) const  [inline]
```

Get the Trust Plugins key establishment algorithm requirements of the DomainParticipant in the context of deriving a shared secret.

8.243 rti::topic::trust::ParticipantTrustProtectionInfo Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Trust Plugins Protection information associated with the discovered DomainParticipant.

```
#include <rti/topic/trust/ParticipantTrustProtectionInfo.hpp>
```

Public Member Functions

- **ParticipantTrustProtectionInfo** ()=default

  Create an instance with the default Trust Plugins Protection Info associated with the DomainParticipant.
- uint32_t **bitmask** () const

  Information about how RTPS wire serialization, discovery, and liveliness interact with the associated Trust Plugins Protection configuration.
- uint32_t **plugin_bitmask** () const

  Internal plugin information that is opaque to DDS.

8.243.1 Detailed Description

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Trust Plugins Protection information associated with the discovered DomainParticipant.

8.243.2 Constructor & Destructor Documentation

8.243.2.1 ParticipantTrustProtectionInfo()

```
rti::topic::trust::ParticipantTrustProtectionInfo::ParticipantTrustProtectionInfo ( )  [default]
```

Create an instance with the default Trust Plugins Protection Info associated with the DomainParticipant.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

8.243.3 Member Function Documentation

8.243.3.1 bitmask()

```
uint32_t rti::topic::trust::ParticipantTrustProtectionInfo::bitmask ( ) const  [inline]
```

Information about how RTPS wire serialization, discovery, and liveliness interact with the associated Trust Plugins Protection configuration.

The meaning of the contents of this field may vary depending on what Trust Plugins the DomainParticipant is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.243.3.2 plugin_bitmask()

```
uint32_t rti::topic::trust::ParticipantTrustProtectionInfo::plugin_bitmask ( ) const  [inline]
```

Internal plugin information that is opaque to DDS.

The meaning of the contents of this field may vary depending on what Trust Plugins the DomainParticipant is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.244 rti::topic::trust::ParticipantTrustSignatureAlgorithmInfo Class Reference

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Trust Plugins signature algorithm information associated with the discovered DomainParticipant.

```
#include <rti/topic/trust/ParticipantTrustSignatureAlgorithmInfo.hpp>
```

Public Member Functions

- **ParticipantTrustSignatureAlgorithmInfo** ()=default

  Create an instance with the default Trust Signature Algorithm Info associated with the discovered DomainParticipant.
- **TrustAlgorithmRequirements trust_chain** () const

  Get the Trust Plugins signature algorithm requirements of the DomainParticipant in the context of creating a chain of trust.
- **TrustAlgorithmRequirements message_auth** () const

  Get the Trust Plugins signature algorithm requirements of the DomainParticipant in the context of checking that messages are authentic.

8.244.1 Detailed Description

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Trust Plugins signature algorithm information associated with the discovered DomainParticipant.

8.244.2 Constructor & Destructor Documentation

8.244.2.1 ParticipantTrustSignatureAlgorithmInfo()

```
rti::topic::trust::ParticipantTrustSignatureAlgorithmInfo::ParticipantTrustSignatureAlgorithmInfo
( )  [default]
```

Create an instance with the default Trust Signature Algorithm Info associated with the discovered DomainParticipant.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

8.244.3 Member Function Documentation

8.244.3.1 trust_chain()

TrustAlgorithmRequirements rti::topic::trust::ParticipantTrustSignatureAlgorithmInfo::trust_chain ( ) const [inline]

Get the Trust Plugins signature algorithm requirements of the DomainParticipant in the context of creating a chain of trust.

8.244.3.2 message_auth()

TrustAlgorithmRequirements rti::topic::trust::ParticipantTrustSignatureAlgorithmInfo::message_↩ auth ( ) const [inline]

Get the Trust Plugins signature algorithm requirements of the DomainParticipant in the context of checking that messages are authentic.

8.245 dds::core::policy::Partition Class Reference

Set of strings that introduces logical partitions in **dds::domain::DomainParticipant** (p. 1326), **dds::pub::Publisher** (p. 1993), or **dds::sub::Subscriber** (p. 2410) entities.

#include <dds/core/policy/CorePolicy.hpp>

Public Member Functions

- **TPartition** ()
  Creates a policy with the default partition.
- **TPartition** (const std::string &partition)
  Creates a policy with a single partition with the specified name.
- **TPartition** (const **dds::core::StringSeq** &partitions)
  Creates a policy with the partitions specified in a vector.
- **TPartition** & **name** (const **dds::core::StringSeq** &partitions)
  Sets the partition names specified in a vector.
- const **dds::core::StringSeq** **name** () const
  Getter (see setter with the same name)

8.245.1 Detailed Description

Set of strings that introduces logical partitions in **dds::domain::DomainParticipant** (p. 1326), **dds::pub::Publisher** (p. 1993), or **dds::sub::Subscriber** (p. 2410) entities.

This QoS policy is used to set string identifiers that are used for partitioning entities that would otherwise be connected to and exchange data with each other:

- A **dds::pub::DataWriter** (p. 1138) within a **dds::pub::Publisher** (p. 1993) only communicates with a **dds::sub::DataReader** (p. 976) in a **dds::sub::Subscriber** (p. 2410) if (in addition to matching the **dds::topic::Topic** (p. 2474) and having compatible QoS) the **dds::pub::Publisher** (p. 1993) and **dds::sub::Subscriber** (p. 2410) have a common partition name string.
- **dds::domain::DomainParticipant** (p. 1326) entities (with the same domain ID and domain tag) are visible to each other only if they have at least one partition name string in common.

Entity:

dds::pub::Publisher (p. 1993), **dds::sub::Subscriber** (p. 2410), **dds::domain::DomainParticipant** (p. 1326)

Properties:

RxO (p. 514) = NO
Changeable (p. 514) = **YES** (p. 514)

8.245.2 Usage

The **Partition** (p. 1923) QoS policy provides another way to control which entities will match-and thus communicate with-which other entities. It can be used to prevent entities that would have otherwise matched from talking to each other. Much in the same way that only applications within the same DDS domain will communicate with each other, only entities that belong to the same partition can talk to each other.

The **Partition** (p. 1923) QoS policy allows you to add one or more strings, "partitions", to an entity:

- A DataWriter and DataReader for the same topic are only considered matched if their Publishers and Subscribers have partitions in common (intersecting partitions).
- DomainParticipants (with the same domain ID and domain tag) are visible to each other only if they have at least one partition in common.

Since the set of partitions for an entity can be dynamically changed, the **Partition** (p. 1923) QoS policy is useful for creating temporary separation groups among entities that would otherwise be connected to and exchange data with each other.

DomainParticipant partitions and Publisher/Subscriber partitions are independent of each other. You can use both features independently or in combination to provide the right level of isolation.

Failure to match partitions is not considered an incompatible QoS and does not trigger any listeners or conditions. A change in this policy *can* potentially modify the "match" of existing DataReader and DataWriter entities. It may establish new "matches" that did not exist before, or break existing matches.

Partition (p. 1923) strings are usually directly matched via string comparisons. However, partition strings can also contain wildcard symbols so that partitions can be matched via pattern matching. As long as the partitions or wildcard patterns of an entity intersect with the partitions or wildcard patterns of otherwise matching entities, the entities match; otherwise they do not.

These partition name patterns are regular expressions as defined by the POSIX fnmatch API (1003.2-1992 section B.↩ 6). A **dds::domain::DomainParticipant** (p. 1326), **dds::pub::Publisher** (p. 1993), or **dds::sub::Subscriber** (p. 2410) entity may include regular expressions in partition names, but no two names that both contain wildcards will ever be considered to match. This means that although regular expressions may be used on the entities, RTI Connext will not try to match two regular expressions.

Each entity must belong to at least one logical partition. A regular expression is not considered to be a logical partition. If an entity has not specified a logical partition, it is assumed to be in the default partition. The default partition is defined to be an empty string (""). Put another way:

- An empty sequence of strings in this QoS policy is considered equivalent to a sequence containing only a single string, the empty string.
- A string sequence that contains only regular expressions and no literal strings, it is treated as if it had an additional element, the empty string.

Partitions are different from creating **dds::core::Entity** (p. 1520) objects in different domains in several ways.

- First, entities belonging to different domains are completely isolated from each other; there is no traffic, meta-traffic or any other way for an application or RTI Connext itself to see entities in a domain it does not belong to.
- Second, a **dds::core::Entity** (p. 1520) can only belong to one domain whereas a **dds::core::Entity** (p. 1520) can be in multiple partitions.
- Finally, as far as RTI Connext is concerned, each unique data instance is identified by the tuple **(DomainID, domain tag, dds::topic::Topic (**p. **2474), key)**. Therefore two **dds::core::Entity** (p. 1520) objects in different domains cannot refer to the same data instance. On the other hand, the same data instance can be made available (published) or requested (subscribed) on one or more partitions.

For more information, see the "PARTITION QosPolicy" section of the `Core Libraries User's Manual`.

8.245.3 Member Function Documentation

8.245.3.1 TPartition() [1/3]

```
dds::core::policy::Partition::TPartition ( )  [inline]
```

Creates a policy with the default partition.

8.245.3.2 TPartition() [2/3]

```
dds::core::policy::Partition::TPartition (
            const std::string & partition )  [inline], [explicit]
```

Creates a policy with a single partition with the specified name.

8.245.3.3 TPartition() [3/3]

```
dds::core::policy::Partition::TPartition (
            const  dds::core::StringSeq & partitions )  [inline], [explicit]
```

Creates a policy with the partitions specified in a vector.

8.245.3.4 name() [1/2]

```
TPartition & dds::core::policy::Partition::name (
            const  dds::core::StringSeq & partitions )  [inline]
```

Sets the partition names specified in a vector.

Several restrictions apply to the partition names in this sequence. A violation of one of the following rules will result in a **dds::core::InconsistentPolicyError** (p. 1610) when setting a **dds::pub::Publisher** (p. 1993)'s or **dds::sub::Subscriber** (p. 2410)'s QoS.

- A partition name string cannot contain the reserved comma character (',').
- The maximum number of partition name strings allowable in a **dds::core::policy::Partition** (p. 1923) is specified on a domain basis in **rti::core::policy::DomainParticipantResourceLimits::max_partitions** (p. 1417). The length of this sequence may not be greater than that value.
- The maximum cumulative length of all partition name strings in a **dds::core::policy::Partition** (p. 1923) is specified on a domain basis in **rti::core::policy::DomainParticipantResourceLimits::max_partition_cumulative_characters** (p. 1417).

[default] Empty sequence (zero-length sequence). Since no logical partition is specified, RTI Connext will assume the entity to be in default partition (empty string partition "").

[range] List of partition name with above restrictions

Categories

Mutable (p. 280), **Discovery Isolation** (p. 183), **Matching** (p. 293), **Discovery Metadata** (p. 288)

8.245.3.5 name() [2/2]

```
const dds::core::StringSeq dds::core::policy::Partition::name ( ) const [inline]
```

Getter (see setter with the same name)

8.246 rti::core::PersistentStorageSettings Class Reference

<<***extension***>> *(*p. *354)* Configures the persistent storage settings for durable writer history and durable reader state.

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **PersistentStorageSettings** ()

  Creates an instance with the default settings.
- **PersistentStorageSettings** & **enable** (bool the_enable)

  <<***extension***>> *(*p. *116) Enables durable writer history in a* ***dds::pub::DataWriter*** *(*p. *1138) and durable reader state in a* ***dds::sub::DataReader*** *(*p. *976).*
- bool **enable** () const

  Getter (see setter with the same name).
- **PersistentStorageSettings** & **file_name** (const **rti::core::optional_value**< std::string > &the_file_name)

  Sets the file name where the durable writer history or durable reader state will be stored.
- **PersistentStorageSettings** & **file_name** (const char *the_file_name)

  Sets the file name where the durable writer history or durable reader state will be stored.
- **rti::core::optional_value**< std::string > **file_name** () const

  Getter (see setter with the same name).
- **PersistentStorageSettings** & **trace_file_name** (const **rti::core::optional_value**< std::string > &the_trace↩ _file_name)

  Sets the file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.
- **PersistentStorageSettings** & **trace_file_name** (const char *the_trace_file_name)

  Sets the file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.
- **rti::core::optional_value**< std::string > **trace_file_name** () const

  Getter (see setter with the same name).
- **PersistentStorageSettings** & **journal_kind** (**rti::core::policy::PersistentJournalKind** the_journal_kind)

  Sets the journal mode of the persistent storage.
- **rti::core::policy::PersistentJournalKind journal_kind** () const

  Getter (see setter with the same name).
- **PersistentStorageSettings** & **synchronization_kind** (**rti::core::policy::PersistentSynchronizationKind** the_synchronization_kind)

  Sets the level of synchronization with the physical disk.
- **rti::core::policy::PersistentSynchronizationKind synchronization_kind** () const

  Getter (see setter with the same name).
- **PersistentStorageSettings** & **vacuum** (bool the_vacuum)

<<***extension***>> *(p. 116) Sets the auto-vacuum status of the storage.*

- bool **vacuum** () const

  Getter (see setter with the same name).

- **PersistentStorageSettings** & **restore** (bool the_restore)

  <<***extension***>> *(p. 116) Indicates if the persisted writer history or reader state must be restored.*

- bool **restore** () const

  Getter (see setter with the same name).

- **PersistentStorageSettings** & **writer_instance_cache_allocation** (const **AllocationSettings** &the_writer_↩ instance_cache_allocation)

  <<***extension***>> *(p. 116) Configures the resource limits associated with the instance durable writer history cache.*

- **AllocationSettings** **writer_instance_cache_allocation** () const

  Gets the writer instance cache allocation (see setter)

- **PersistentStorageSettings** & **writer_sample_cache_allocation** (const **AllocationSettings** &the_writer_↩ sample_cache_allocation)

  <<***extension***>> *(p. 116) Configures the resource limits associated with the sample durable writer history cache.*

- **AllocationSettings** **writer_sample_cache_allocation** () const

  Gets the writer instance cache allocation (see setter)

- **PersistentStorageSettings** & **writer_memory_state** (bool the_writer_memory_state)

  <<***extension***>> *(p. 116) Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.*

- bool **writer_memory_state** () const

  Getter (see setter with the same name).

- **PersistentStorageSettings** & **reader_checkpoint_frequency** (uint32_t the_reader_checkpoint_frequency)

  <<***extension***>> *(p. 116) Controls how often the reader state is stored into the database.*

- uint32_t **reader_checkpoint_frequency** () const

  Getter (see setter with the same name).

8.246.1 Detailed Description

<<***extension***>> *(p. 354)* Configures the persistent storage settings for durable writer history and durable reader state.

In a **dds::pub::DataWriter** (p. 1138), this structure configures durable writer history. This feature allows a DataWriter to persist its historical cache, so that it can survive shutdowns, crashes, and restarts. When an application restarts, each DataWriter that has been configured to have durable writer history automatically loads all of the data in this cache from disk and can carry on sending data as if it had never stopped executing.

In a **dds::sub::DataReader** (p. 976), this structure configures durable reader state. This feature allows a DataReader to persist its state and remember which data it has already received. When an application restarts, each DataReader that has been configured to have durable reader state automatically loads its state from disk and can carry on receiving data as if it had never stopped executing. Data that had already been received by the DataReader before the restart will be suppressed so that it is not even sent over the network.

RTI Connext uses SQLite to store the durable writer history and durable reader state.

QoS:

dds::core::policy::Durability (p. 1433)

8.246.2 Constructor & Destructor Documentation

8.246.2.1 PersistentStorageSettings()

```
rti::core::PersistentStorageSettings::PersistentStorageSettings ( )  [inline]
```

Creates an instance with the default settings.

8.246.3 Member Function Documentation

8.246.3.1 enable() [1/2]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::enable (
            bool the_enable )
```

<<***extension***>> *(p. 116)* Enables durable writer history in a **dds::pub::DataWriter** (p. 1138) and durable reader state in a **dds::sub::DataReader** (p. 976).

When this field is set to true, the persistent storage configuration using this structure will take precedence over the configuration using the deprecated **dds.data_writer.history.odbc_plugin.builtin**.∗ and **dds.data_reader.state**.∗ properties.

[default] false

Categories

Immutable (p. 243), **Durability** (p. 184)

8.246.3.2 enable() [2/2]

```
bool rti::core::PersistentStorageSettings::enable ( ) const
```

Getter (see setter with the same name).

8.246.3.3 file_name() [1/3]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::file_name (
            const  rti::core::optional_value< std::string > & the_file_name )
```

Sets the file name where the durable writer history or durable reader state will be stored.

Parameters

the_file_name	An optional string. A value is required when enable is set to true.

Setting this field to a value other than NULL is mandatory when enabling durable writer history or durable reader state.

If the file does not exist, it will be created.

If the file exists and **rti::core::PersistentStorageSettings::restore** (p. 1934) is set to true, the durable writer history or durable reader state will be restored from the file. Otherwise, the file will be overwritten.

Important: When the file exists, the fields **rti::core::policy::DataReaderProtocol::virtual_guid** (p. 1060) and **rti↩ ::core::policy::DataWriterProtocol::virtual_guid** (p. 1212) will be set by RTI Connext based on the file content. If you change these fields, the value will be ignored.

RTI Connext uses SQLite to store the durable writer history and durable reader state.

[default] NULL

Categories

Immutable (p. 243), **Durability** (p. 184)

8.246.3.4 file_name() [2/3]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::file_name (
            const char * the_file_name )
```

Sets the file name where the durable writer history or durable reader state will be stored.

Parameters

the_file_name	The file name. A value different other than NULL is required when enable is set to true.

Setting this field to a value other than NULL is mandatory when enabling durable writer history or durable reader state.

If the file does not exist, it will be created.

If the file exists and **rti::core::PersistentStorageSettings::restore** (p. 1934) is set to true, the durable writer history or durable reader state will be restored from the file. Otherwise, the file will be overwritten.

Important: When the file exists, the fields **rti::core::policy::DataReaderProtocol::virtual_guid** (p. 1060) and **rti↩ ::core::policy::DataWriterProtocol::virtual_guid** (p. 1212) will be set by RTI Connext based on the file content. If you change these fields, the value will be ignored.

RTI Connext uses SQLite to store the durable writer history and durable reader state.

[default] NULL

Categories

Immutable (p. 243), **Durability** (p. 184)

8.246.3.5 file_name() [3/3]

```
rti::core::optional_value< std::string > rti::core::PersistentStorageSettings::file_name ( )
const
```

Getter (see setter with the same name).

8.246.3.6 trace_file_name() [1/3]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::trace_file_name (
            const  rti::core::optional_value< std::string > & the_trace_file_name )
```

Sets the file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.

Parameters

the_trace_file_name	An optional string. When not set, no trace file will be generated.

Setting this field to a value other than NULL will enable tracing of the SQL statements executed when loading and storing the durable writer history or durable reader state.

Important: Enabling tracing will have a negative impact on performance. Use this feature only for debugging purposes.

[default] NULL

Categories

Immutable (p. 243), **Durability** (p. 184)

8.246.3.7 trace_file_name() [2/3]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::trace_file_name (
            const char * the_trace_file_name )
```

Sets the file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.

Parameters

the_trace_file_name	The trace file name. When NULL, no trace file will be generated.

Setting this field to a value other than NULL will enable tracing of the SQL statements executed when loading and storing the durable writer history or durable reader state.

Important: Enabling tracing will have a negative impact on performance. Use this feature only for debugging purposes.

[default] NULL

Categories

Immutable (p. 243), **Durability** (p. 184)

8.246.3.8 trace_file_name() [3/3]

```
rti::core::optional_value< std::string > rti::core::PersistentStorageSettings::trace_file_name (
) const
```

Getter (see setter with the same name).

8.246.3.9 journal_kind() [1/2]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::journal_kind (
            rti::core::policy::PersistentJournalKind the_journal_kind )
```

Sets the journal mode of the persistent storage.

Parameters

the_journal_kind	The journal kind.

[default] rti::core::policy::PersistentJournalKind::wal_journal (p. 530)

Categories

Immutable (p. 243), **Durability** (p. 184)

8.246.3.10 journal_kind() [2/2]

```
rti::core::policy::PersistentJournalKind rti::core::PersistentStorageSettings::journal_kind ( )
const
```

Getter (see setter with the same name).

8.246.3.11 synchronization_kind() [1/2]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::synchronization_kind (
            rti::core::policy::PersistentSynchronizationKind the_synchronization_kind )
```

Sets the level of synchronization with the physical disk.

Parameters

the_synchronization_kind	The synchronization kind.

[default] rti::core::policy::PersistentSynchronizationKind::normal (p. 531)

Categories

Immutable (p. 243), **Durability** (p. 184)

8.246.3.12 synchronization_kind() [2/2]

```
rti::core::policy::PersistentSynchronizationKind rti::core::PersistentStorageSettings::synchronization↩
_kind ( ) const
```

Getter (see setter with the same name).

8.246.3.13 vacuum() [1/2]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::vacuum (
            bool the_vacuum )
```

<<***extension***>> *(p. 116)* Sets the auto-vacuum status of the storage.

When auto-vacuum is true, the storage files will be compacted automatically with every transaction.

When auto-vacuum is false, after data is deleted from the storage files, the files remain the same size.

[default] true

Categories

Immutable (p. 243), **Durability** (p. 184)

8.246.3.14 vacuum() [2/2]

```
bool rti::core::PersistentStorageSettings::vacuum ( ) const
```

Getter (see setter with the same name).

8.246.3.15 restore() [1/2]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::restore (
            bool the_restore )
```

<<***extension***>> *(*p. *116)* Indicates if the persisted writer history or reader state must be restored.

For a **dds::pub::DataWriter** (p. 1138), this field indicates whether or not the persisted writer history must be restored once the DataWriter is restarted.

For a **dds::sub::DataReader** (p. 976), this field indicates whether or not the persisted reader state must be restored once the DataReader is restarted.

[default] true

Categories

Immutable (p. 243), **Durability** (p. 184)

8.246.3.16 restore() [2/2]

```
bool rti::core::PersistentStorageSettings::restore ( ) const
```

Getter (see setter with the same name).

8.246.3.17 writer_instance_cache_allocation() [1/2]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::writer_instance_cache_allocation
(
            const  AllocationSettings & the_writer_instance_cache_allocation )
```

<<***extension***>> *(p. 116)* Configures the resource limits associated with the instance durable writer history cache.

This field only applies to **dds::pub::DataWriter** (p. 1138) entities.

To minimize the number of accesses to the persisted storage, RTI Connext uses an instance cache.

Do not confuse this limit with the initial and maximum number of instances that can be maintained by a DataWriter in persistent storage. These resource limits are configured using **dds::core::policy::ResourceLimits::max_instances** (p. 2206) and **dds::core::policy::ResourceLimits::initial_instances** (p. 2208).

If **rti::core::PersistentStorageSettings::writer_memory_state** (p. 1936) is set to true, then the value of **rti::core::AllocationSettings::max_count** (p. 804) is set to **dds::core::LENGTH_UNLIMITED** (p. 438), overwriting any value set by the user.

rti::core::AllocationSettings::incremental_count (p. 805) is ignored.

[range] rti::core::AllocationSettings::max_count (p. 804) in interval [1, INT_MAX], rti::core::LENGTH_AUTO, or **dds::core::LENGTH_UNLIMITED** (p. 438). **rti::core::AllocationSettings::initial_count** (p. 804) in interval [1, INT_MAX], or rti::core::LENGTH_AUTO.

rti::core::LENGTH_AUTO means that the value will be set to the equivalent value of **dds::core::policy::ResourceLimits** (p. 2203).

[default] rti::core::AllocationSettings::max_count (p. 804) = rti::core::LENGTH_AUTO (= **dds::core::policy::ResourceLimits::max_instances** (p. 2206)) and **rti::core::AllocationSettings::initial_count** (p. 804) = rti::core::LENGTH_AUTO (= **dds::core::policy::ResourceLimits::initial_instances** (p. 2208)).

Categories

Immutable (p. 243), **Durability** (p. 184), **Sample Memory Management** (p. 230)

8.246.3.18 writer_instance_cache_allocation() [2/2]

```
AllocationSettings rti::core::PersistentStorageSettings::writer_instance_cache_allocation ( )
const
```

Gets the writer instance cache allocation (see setter)

8.246.3.19 writer_sample_cache_allocation() [1/2]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::writer_sample_cache_allocation
(
            const  AllocationSettings & the_writer_sample_cache_allocation )
```

<<***extension***>> *(p. 116)* Configures the resource limits associated with the sample durable writer history cache.

This field only applies to **dds::pub::DataWriter** (p. 1138) entities.

To minimize the number of accesses to the persisted storage, RTI Connext uses a sample cache.

Do not confuse this limit with the initial and maximum number of samples that can be maintained by a DataWriter in persistent storage. These resource limits are configured using **dds::core::policy::ResourceLimits::max_samples** (p. 2205) and **dds::core::policy::ResourceLimits::initial_samples** (p. 2207).

rti::core::AllocationSettings::incremental_count (p. 805) is ignored.

[range] rti::core::AllocationSettings::max_count (p. 804) in interval [1, INT_MAX], rti::core::LENGTH_AUTO, or **dds::core::LENGTH_UNLIMITED** (p. 438). **rti::core::AllocationSettings::initial_count** (p. 804) in interval [1, INT_↩ MAX], or rti::core::LENGTH_AUTO.

rti::core::LENGTH_AUTO means that the value will be set to the equivalent value of **dds::core::policy::Resource↩ Limits** (p. 2203).

[default] rti::core::AllocationSettings::max_count (p. 804) = 32 and **rti::core::AllocationSettings::initial_count** (p. 804) = 32.

Categories

Immutable (p. 243), **Durability** (p. 184), **Sample Memory Management** (p. 230)

8.246.3.20 writer_sample_cache_allocation() [2/2]

```
AllocationSettings rti::core::PersistentStorageSettings::writer_sample_cache_allocation ( ) const
```

Gets the writer instance cache allocation (see setter)

8.246.3.21 writer_memory_state() [1/2]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::writer_memory_state (
            bool the_writer_memory_state )
```

<<***extension***>> *(p. 116)* Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.

This field only applies to **dds::pub::DataWriter** (p. 1138) entities.

If this field is set to true, then **rti::core::AllocationSettings::max_count** (p. 804) in **rti::core::PersistentStorage↩ Settings::writer_instance_cache_allocation** (p. 1934) is set to **dds::core::LENGTH_UNLIMITED** (p. 438), overwriting any value set by the user.

In addition, the durable writer history will keep a fixed state overhead per sample in memory. This mode provides the best durable writer history performance. However, the restore operation will be slower, and the maximum number of samples that the durable writer history can manage is limited by the available physical memory.

If this field is set to false, all the state will be kept in the underlying database. In this mode, the maximum number of samples in the durable writer history is not limited by the physical memory available.

This field is always set to false when the DataWriter is configured with **dds::core::policy::Reliability↩ ::acknowledgment_kind** (p. 2154) set to **rti::core::policy::AcknowledgmentKind_def::APPLICATION_AUTO** (p. 798) or **rti::core::policy::AcknowledgmentKind_def::APPLICATION_EXPLICIT** (p. 798), or **rti::core::policy::↩ Availability::enable_required_subscriptions** (p. 871) is set to true.

[default] true

Categories

Immutable (p. 243), **Durability** (p. 184), **Middleware Memory Management** (p. 204)

8.246.3.22 writer_memory_state() [2/2]

```
bool rti::core::PersistentStorageSettings::writer_memory_state ( ) const
```

Getter (see setter with the same name).

8.246.3.23 reader_checkpoint_frequency() [1/2]

```
PersistentStorageSettings & rti::core::PersistentStorageSettings::reader_checkpoint_frequency (
            uint32_t the_reader_checkpoint_frequency )
```

<<***extension***>> *(p. 116)* Controls how often the reader state is stored into the database.

This field only applies to **dds::sub::DataReader** (p. 976) entities.

A value of N means store the state once every N received and processed samples.

The circumstances under which a data sample is considered "processed by the application" depends on the DataReader configuration.

For additional information on when a sample is considered "processed by the application" see `Durable Reader State`, in the Core Libraries User's Manual.

A high value will provide better performance. However, if the DataReader is restarted it may receive some duplicate samples.

[range] [1, 1000000] **[default]** 1

Categories

Immutable (p. 243), **Durability** (p. 184)

8.246.3.24 reader_checkpoint_frequency() [2/2]

```
uint32_t rti::core::PersistentStorageSettings::reader_checkpoint_frequency ( ) const
```

Getter (see setter with the same name).

8.247 rti::core::pointer< T > Class Template Reference

`#include <Pointer.hpp>`

Public Member Functions

- **pointer** () OMG_NOEXCEPT
- **pointer** (T *ptr) OMG_NOEXCEPT
- void **set** (T *ptr) OMG_NOEXCEPT
- T * **get** () const OMG_NOEXCEPT
- **pointer**< T > & **operator=** (T *other) OMG_NOEXCEPT
- bool **operator==** (const **pointer**< T > &other) const
- bool **operator!=** (const **pointer**< T > &other) const

8.247.1 Detailed Description

template<typename T>
class rti::core::pointer< T >

Helper class for generated types that contain IDL pointers (an RTI extension)

It wraps a loaned pointer and provides overloads of certain operators to access the contents rather than the pointer. It never deep-copies or deletes the underlying pointer.

8.247.2 Constructor & Destructor Documentation

8.247.2.1 pointer() [1/2]

```
template<typename T >
rti::core::pointer< T >::pointer ( )  [inline]
```

Initialize to NULL

8.247.2.2 pointer() [2/2]

```
template<typename T >
rti::core::pointer< T >::pointer (
            T * ptr )  [inline]
```

Shallow-copy the pointer and allow automatic conversion from T *

8.247.3 Member Function Documentation

8.247.3.1 set()

```
template<typename T >
void  rti::core::pointer< T >::set (
            T * ptr )  [inline]
```

Set the underlying pointer. This function does not copy or delete, it just replaces this pointer.

8.247.3.2 get()

```
template<typename T >
T *  rti::core::pointer< T >::get ( ) const  [inline]
```

Returns the underlying pointer

Referenced by **rti::core::pointer< T >::operator==()**.

8.247.3.3 operator=()

```
template<typename T >
pointer< T > &  rti::core::pointer< T >::operator= (
            T * other )  [inline]
```

Assign another pointer replacing the current one

8.247.3.4 operator==()

```
template<typename T >
bool  rti::core::pointer< T >::operator== (
            const  pointer< T > & other ) const  [inline]
```

Deep equality comparison

References **rti::core::pointer< T >::get()**.

8.247.3.5 operator"!=()

```
template<typename T >
bool  rti::core::pointer< T >::operator!= (
            const  pointer< T > & other ) const  [inline]
```

Deep inequality comparison

8.248 dds::core::policy::policy_id< Policy > Class Template Reference

Obtains the QosPolicyId of a QoS Policy.

`#include <PolicyTraits.hpp>`

8.248.1 Detailed Description

template<typename Policy>
class dds::core::policy::policy_id< Policy >

Obtains the QosPolicyId of a QoS Policy.

Template Parameters

A	QoS policy from the namespaces **dds::core::policy** (p. 627) or rti::core::policy

Specializations for each QoS policy provide the following constant:

```
static const dds::core::policy::QosPolicyId value;
```

For example, the policy ID for the **dds::core::policy::Deadline** (p. 1255) policy is `dds::core::policy::policy_id<dds::core::policy::Deadline>::value`.

8.249 dds::core::policy::policy_name< Policy > Class Template Reference

Obtains the policy name.

```
#include <PolicyTraits.hpp>
```

8.249.1 Detailed Description

template<typename Policy>
class dds::core::policy::policy_name< Policy >

Obtains the policy name.

Template Parameters

A	QoS policy from the namespaces **dds::core::policy** (p. 627) or rti::core::policy

Specializations for each QoS policy provide the following method:

```
static const std::string& name();
```

For example:

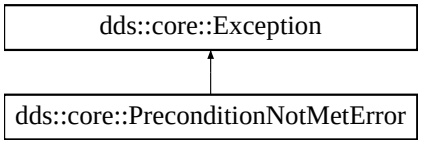
```
using namespace dds::core::policy;
std::cout « policy_name<Reliability>::name() « std::endl; // Prints "Reliability"
```

8.250 dds::core::PreconditionNotMetError Class Reference

A **PreconditionNotMetError** (p. 1941) is thrown when a pre-condition for the operation was not met.

```
#include <Exception.hpp>
```

Inheritance diagram for dds::core::PreconditionNotMetError:



Public Member Functions

- virtual const char * **what** () const throw ()

  *Access the message contained in this **PreconditionNotMetError** (p. 1941) exception.*

8.250.1 Detailed Description

A **PreconditionNotMetError** (p. 1941) is thrown when a pre-condition for the operation was not met.

Inherits also from `std::logic_error`

The system is not in the expected state when the function is called, or the parameter itself is not in the expected state when the function is called.

8.250.2 Member Function Documentation

8.250.2.1 what()

```
virtual const char * dds::core::PreconditionNotMetError::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **PreconditionNotMetError** (p. 1941) exception.

Returns

The message.

Implements **dds::core::Exception** (p. 1547).

8.251 dds::core::policy::Presentation Class Reference

Specifies how the samples representing changes to data instances are presented to a subscribing application.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **Presentation** ()

  Create an instance with the default value.
- **Presentation** (**dds::core::policy::PresentationAccessScopeKind** the_access_scope, bool the_coherent_↩ access, bool the_ordered_access)

  Creates an instance with the specified access scope and coherent and ordered access.
- **Presentation** & **access_scope** (**dds::core::policy::PresentationAccessScopeKind** the_access_scope)

  Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
- **dds::core::policy::PresentationAccessScopeKind** **access_scope** () const

  Getter (see setter with the same name)
- **Presentation** & **coherent_access** (bool enable)

  Controls whether coherent access is supported within the access_scope.
- bool **coherent_access** () const

  Getter (see setter with the same name)
- **Presentation** & **ordered_access** (bool enable)

  Controls whether ordered access is supported within the access_scope.
- bool **ordered_access** () const

  Getter (see setter with the same name)
- **dds::core::policy::Presentation** & **drop_incomplete_coherent_set** (bool enable)

  <<***extension***>> (p. *116) Indicates whether or not a DataReader should drop samples from an incomplete coherent set for which not all the relevant samples where received.*
- bool **drop_incomplete_coherent_set** () const

  <<***extension***>> (p. *116) Getter (see setter with the same name)*

Static Public Member Functions

- static **Presentation** **GroupAccessScope** (bool coherent, bool ordered)

  Creates a ***Presentation*** (p. *1942) instnace with group access scope.*
- static **Presentation** **InstanceAccessScope** (bool coherent, bool ordered)

  Creates a ***Presentation*** (p. *1942) instance with instance access scope.*
- static **Presentation** **TopicAccessScope** (bool coherent, bool ordered)

  Creates a ***Presentation*** (p. *1942) instance with topic access scope.*

8.251.1 Detailed Description

Specifies how the samples representing changes to data instances are presented to a subscribing application.

This QoS policy controls the extent to which changes to data instances can be made dependent on each other and also the kind of dependencies that can be propagated and maintained by RTI Connext. Specifically, this policy affects the application's ability to:

- specify and receive coherent changes to instances
- specify the relative order in which changes are presented

Entity:

dds::pub::Publisher (p. 1993), **dds::sub::Subscriber** (p. 2410)

Status:

dds::core::status::StatusMask::offered_incompatible_qos() (p. 2380), **dds::core::status::StatusMask↩ ::requested_incompatible_qos()** (p. 2381)

Properties:

RxO (p. 514) = YES
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

8.251.2 Usage

A **dds::sub::DataReader** (p. 976) will usually receive data in the order that it was sent by a **dds::pub::DataWriter** (p. 1138), and the data is presented to the **dds::sub::DataReader** (p. 976) as soon as the application receives the next expected value. However, sometimes you may want a set of data for the same DataWriter or different DataWriters to be presented to the DataReader(s) only after *all* of the elements of the set have been received, but not before. Or you may want the data to be presented in a different order than that in which it was sent. Specifically for keyed data, you may want the middleware to present the data in keyed – or *instance* – order, such that samples pertaining to the same instance are presented together.

The **Presentation** (p. 1942) QoS policy is a request-offered QoS policy that allows you to specify different *scopes* of presentation. It also controls whether or not a set of changes within the scope is delivered at the same time or can be delivered as soon as each element is received.

- `coherent_access` controls whether RTI Connext will preserve the groupings of changes made by a publishing application by means of the operations **dds::pub::CoherentSet::CoherentSet** (p. 932) and **dds::pub::↩ CoherentSet::end()** (p. 933).

- `ordered_access` controls whether RTI Connext will preserve the order of changes.

- `access_scope` controls the granularity of the other settings. See below:

If `coherent_access` is set, then the `access_scope` set on the **dds::sub::Subscriber** (p. 2410) controls the maximum extent of coherent changes. The behavior is as follows:

- If `access_scope` is set to **dds::core::policy::PresentationAccessScopeKind_def::INSTANCE** (p. 1951) (the default), the behavior is the same as **dds::core::policy::PresentationAccessScopeKind_def::TOPIC** (p. 1951).

- If access_scope is set to **dds::core::policy::PresentationAccessScopeKind_def::TOPIC** (p. 1951), then coherent changes (indicated by their enclosure within calls to **dds::pub::CoherentSet::CoherentSet** (p. 932) and **dds::pub::CoherentSet::end()** (p. 933)) will be made available as a unit to each remote **dds::sub::DataReader** (p. 976) independently. That is, changes made to instances within each individual **dds::pub::DataWriter** (p. 1138) will be presented as a unit. They will not be grouped with changes made to instances belonging to a different **dds::pub::DataWriter** (p. 1138).

- If access_scope is set to **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951), then coherent changes made to instances through a set of **dds::pub::DataWriter** (p. 1138) entities attached to a common **dds::pub::Publisher** (p. 1993) will be made available as a unit to the **dds::sub::DataReader** (p. 976) entities within a **dds::sub::Subscriber** (p. 2410).

If ordered_access is set, then the access_scope set on the **dds::sub::Subscriber** (p. 2410) controls the maximum extent to which order will be preserved by RTI Connext.

- If access_scope is set to **dds::core::policy::PresentationAccessScopeKind_def::INSTANCE** (p. 1951) (the lowest level), then the relative order of DDS samples sent by a **dds::pub::DataWriter** (p. 1138) is only preserved on a per-instance basis. If two DDS samples refer to the same instance (identified by Topic and a particular value for the key), then the order in which they are stored in the **dds::sub::DataReader** (p. 976) queue is consistent with the order in which the changes occurred. However, if the two DDS samples belong to different instances, the order in which they are presented may or may not match the order in which the changes occurred. This is the case even if it is the same application thread making the changes using the same **dds::pub::DataWriter** (p. 1138).

- If access_scope is set to **dds::core::policy::PresentationAccessScopeKind_def::TOPIC** (p. 1951), changes (creations, deletions, modifications) made by a single **dds::pub::DataWriter** (p. 1138) to one or more instances are presented to a **dds::sub::DataReader** (p. 976) in the same order in which they occur. Changes made to instances though different **dds::pub::DataWriter** (p. 1138) entities are not required to be presented in the order in which they occur. This is the case even if the changes are made by a single application thread using **dds::pub::DataWriter** (p. 1138) objects attached to the same **dds::pub::Publisher** (p. 1993).

- Finally, if access_scope is set to **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951), changes made to instances via **dds::pub::DataWriter** (p. 1138) entities attached to the same **dds::pub::Publisher** (p. 1993) object are presented to the **dds::sub::DataReader** (p. 976) entities within a **dds::sub::Subscriber** (p. 2410) in the same order in which they occur. For this to happen, the application must take the samples using the Subscriber's **dds::sub::CoherentAccess::CoherentAccess()** (p. 930) and **dds::sub::CoherentAccess::end()** (p. 931) operations.

See the "Ordered Access" section in the "PRESENTATION QosPolicy" section of the Core Libraries User's Manual for more information.

If access_scope is set to **dds::core::policy::PresentationAccessScopeKind_def::HIGHEST_OFFERED** (p. 1951) on the **dds::sub::Subscriber** (p. 2410), then the **dds::sub::Subscriber** (p. 2410) will use the access_scope offered by the remote **dds::pub::Publisher** (p. 1993).

Note that this QoS policy controls the scope at which related changes are made available to the subscriber. This means the subscriber **can** access the changes in a coherent manner and in the proper order; however, it does not necessarily imply that the **dds::sub::Subscriber** (p. 2410) **will** indeed access the changes in the correct order. For that to occur, the application at the subscriber end must use the proper logic in reading the **dds::sub::DataReader** (p. 976) objects.

For **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951) the subscribing application must use the APIs **dds::sub::CoherentAccess::CoherentAccess()** (p. 930), **dds::sub::CoherentAccess::end()** (p. 931) and **dds::sub::find** (p. 672) to access the changes in the proper order. If you do not use these operations, the data may not be ordered across DataWriters that belong to the same **dds::pub::Publisher** (p. 1993).

The field **dds::sub::SampleInfo::coherent_set_info** (p. 2295) is set when a sample is part of a coherent set. This field provides information to identify the coherent set that a sample is part of. In addition, **rti::core::CoherentSetInfo::incomplete_coherent_set** (p. 938) indicates if a sample is part of an incomplete coherent set (one for which not all samples have been received). Coherent sets for which some of the samples are filtered out by content or time on the **dds::pub::DataWriter** (p. 1138) are considered incomplete.

By default, the samples that are received from an incomplete coherent set are dropped by the DataReader(s) and they are not provided to the application. Such samples are reported as lost in the SAMPLE_LOST Status. By setting **dds::core::policy::Presentation::drop_incomplete_coherent_set** (p. 1949) to false, you can change this behavior and, in this case, samples from incomplete coherent sets will be provided to the application. These samples have **rti::core::CoherentSetInfo::incomplete_coherent_set** (p. 938) set to true.

8.251.3 Compatibility

The value offered is considered compatible with the value requested if and only if the following conditions are met:

- the inequality *offered* `access_scope` >= *requested* `access_scope` evaluates to 'TRUE' or requested `access_scope` is **dds::core::policy::PresentationAccessScopeKind_def::HIGHEST_OFFERED** (p. 1951). For the purposes of this inequality, the values of `access_scope` are considered ordered such that **dds::core::policy::PresentationAccessScopeKind_def::INSTANCE** (p. 1951) < **dds::core::policy::PresentationAccessScopeKind_def::TOPIC** (p. 1951) < **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951).

- requested `coherent_access` is false, or else both offered and requested `coherent_access` are true.

- requested `ordered_access` is false, or else both offered and requested `ordered_access` are true.

8.251.4 Constructor & Destructor Documentation

8.251.4.1 Presentation() [1/2]

```
dds::core::policy::Presentation::Presentation ( )  [inline]
```

Create an instance with the default value.

8.251.4.2 Presentation() [2/2]

```
dds::core::policy::Presentation::Presentation (
             dds::core::policy::PresentationAccessScopeKind the_access_scope,
            bool the_coherent_access,
            bool the_ordered_access )  [inline]
```

Creates an instance with the specified access scope and coherent and ordered access.

See the corresponding setters.

8.251.5 Member Function Documentation

8.251.5.1 access_scope() [1/2]

```
Presentation & dds::core::policy::Presentation::access_scope (
            dds::core::policy::PresentationAccessScopeKind the_access_scope )  [inline]
```

Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.

[default] dds::core::policy::PresentationAccessScopeKind_def::INSTANCE (p. 1951)

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Applied Per-Instance** (p. 287), **Sample Ordering** (p. 234), **Data Availability** (p. 167)

8.251.5.2 access_scope() [2/2]

```
dds::core::policy::PresentationAccessScopeKind dds::core::policy::Presentation::access_scope ( )
const  [inline]
```

Getter (see setter with the same name)

8.251.5.3 coherent_access() [1/2]

```
Presentation & dds::core::policy::Presentation::coherent_access (
            bool enable )  [inline]
```

Controls whether coherent access is supported within the access_scope.

That is, the ability to group a set of changes as a unit on the publishing end such that they are received as a unit at the subscribing end.

Note: To use this feature, the DataWriter must be configured for RELIABLE communication (see **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157)).

[default] false

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Sample Ordering** (p. 234), **Data Availability** (p. 167)

8.251.5.4 coherent_access() [2/2]

```
bool dds::core::policy::Presentation::coherent_access ( ) const  [inline]
```

Getter (see setter with the same name)

8.251.5.5 ordered_access() [1/2]

```
Presentation & dds::core::policy::Presentation::ordered_access (
            bool enable )  [inline]
```

Controls whether ordered access is supported within the access_scope.

That is, the ability of the subscriber to see changes in the same order as they occurred on the publishing end.

[default] false

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Applied Per-Instance** (p. 287), **Sample Ordering** (p. 234), **Data Availability** (p. 167)

8.251.5.6 ordered_access() [2/2]

```
bool dds::core::policy::Presentation::ordered_access ( ) const  [inline]
```

Getter (see setter with the same name)

8.251.5.7 GroupAccessScope()

```
static  Presentation dds::core::policy::Presentation::GroupAccessScope (
            bool coherent,
            bool ordered )  [inline], [static]
```

Creates a **Presentation** (p. 1942) instnace with group access scope.

8.251.5.8 InstanceAccessScope()

```
static  Presentation dds::core::policy::Presentation::InstanceAccessScope (
            bool coherent,
            bool ordered )  [inline], [static]
```

Creates a **Presentation** (p. 1942) instance with instance access scope.

8.251.5.9 TopicAccessScope()

```
static  Presentation dds::core::policy::Presentation::TopicAccessScope (
            bool coherent,
            bool ordered )  [inline], [static]
```

Creates a **Presentation** (p. 1942) instance with topic access scope.

8.251.5.10 drop_incomplete_coherent_set() [1/2]

```
dds::core::policy::Presentation & drop_incomplete_coherent_set (
            bool enable )
```

<<***extension***>> *(p. 116)* Indicates whether or not a DataReader should drop samples from an incomplete coherent set for which not all the relevant samples where received.

Note that a coherent set will be considered incomplete if some of its samples are filtered by content or time on the DataWriter side.

By default, the samples that are received from an incomplete coherent set are dropped (and reported as lost) by the DataReader(s) and they are not provided to the application. By setting this parameter to false, you can change this behavior.

Samples from an incomplete coherent set have **rti::core::CoherentSetInfo::incomplete_coherent_set** (p. 938) set to true in **dds::sub::SampleInfo::coherent_set_info** (p. 2295).

[default] true

Categories

Immutable (p. 243), **Sample Ordering** (p. 234), **Data Availability** (p. 167)

8.251.5.11 drop_incomplete_coherent_set() [2/2]

```
bool drop_incomplete_coherent_set ( ) const
```

<<***extension***>> *(p. 116)* Getter (see setter with the same name)

8.252 dds::core::policy::PresentationAccessScopeKind_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) PresentationAccessScopeKind.

```
#include <PolicyKind.hpp>
```

Public Types

- enum **type** {
  INSTANCE ,
  TOPIC ,
  GROUP ,
  HIGHEST_OFFERED }

  The underlying `enum` *type.*

8.252.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) PresentationAccessScopeKind.

8.252.2 Member Enumeration Documentation

8.252.2.1 type

`enum` **dds::core::policy::PresentationAccessScopeKind_def::type**

The underlying `enum` type.

Enumerator

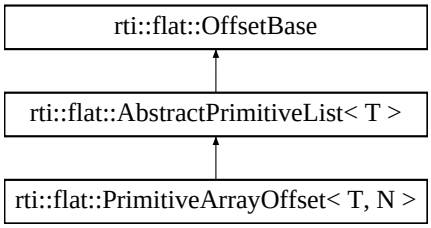
INSTANCE	**[default]** Scope spans only a single instance. Indicates that changes to one instance need not be coherent nor ordered with respect to changes to any other instance. In other words, order and coherent changes apply to each instance separately.
TOPIC	Scope spans all instances within the same **dds::pub::DataWriter** (p. 1138), but not across instances in different **dds::pub::DataWriter** (p. 1138) entities.
GROUP	Scope spans all instances belonging to **dds::pub::DataWriter** (p. 1138) entities within the same **dds::pub::Publisher** (p. 1993).
HIGHEST_OFFERED	This value only applies to a **dds::sub::Subscriber** (p. 2410). The **dds::sub::Subscriber** (p. 2410) will use the access scope specified by each remote **dds::pub::Publisher** (p. 1993).

8.253 rti::flat::PrimitiveArrayOffset< T, N > Class Template Reference

Offset to an array of primitive elements.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::PrimitiveArrayOffset< T, N >:



Public Member Functions

- unsigned int **element_count** () const

  Returns the number of elements, N.

8.253.1 Detailed Description

template<typename T, unsigned int N>
class rti::flat::PrimitiveArrayOffset< T, N >

Offset to an array of primitive elements.

Template Parameters

T	The primitive type
N	The array bound. For multidimensional arrays, N is the product of each dimension bound.

A **PrimitiveArrayOffset** (p. 1951) may meet the requirements to be cast to an array of the equivalent plain C++ element type (see **rti::flat::plain_cast()** (p. 417)).

8.253.2 Member Function Documentation

8.253.2.1 element_count()

```
template<typename T , unsigned int N>
unsigned int  rti::flat::PrimitiveArrayOffset< T, N >::element_count ( ) const  [inline]
```

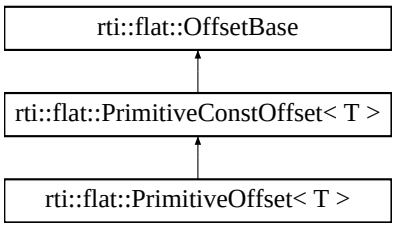
Returns the number of elements, N.

8.254 rti::flat::PrimitiveConstOffset< T > Struct Template Reference

A const Offset to an optional primitive member.

`#include <Offset.hpp>`

Inheritance diagram for rti::flat::PrimitiveConstOffset< T >:



Public Member Functions

- T **get** () const

  Gets the value of this primitive member.

8.254.1 Detailed Description

template<typename T>
struct rti::flat::PrimitiveConstOffset< T >

A const Offset to an optional primitive member.

Read-only version of **PrimitiveOffset** (p. 1953).

If !is_null(), the value can be retrieved with **get()** (p. 1953).

8.254.2 Member Function Documentation

8.254.2.1 get()

```
template<typename T >
T  rti::flat::PrimitiveConstOffset< T >::get ( ) const  [inline]
```

Gets the value of this primitive member.

Precondition

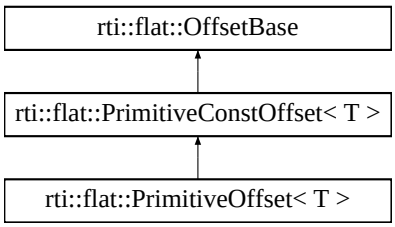
!is_null()

8.255 rti::flat::PrimitiveOffset< T > Struct Template Reference

An Offset to an optional primitive member.

```
#include <Offset.hpp>
```

Inheritance diagram for rti::flat::PrimitiveOffset< T >:



Public Member Functions

- bool **set** (T value)

  Sets a value for this primitive member.

8.255.1 Detailed Description

template<typename T>
struct rti::flat::PrimitiveOffset< T >

An Offset to an optional primitive member.

Template Parameters

T	The primitive type

Non-optional primitive members are accessed directly using its container type's methods; however, since an optional member may not exist, this Offset is returned instead. The only purpose of the type **PrimitiveOffset** (p. 1953) is to provide a way to check for the member's existence.

If !is_null(), the value can be retrieved with **get()** (p. 1953) or set with **set()** (p. 1954).

See also

MyFlatMutableOffset::my_optional_primitive (p. 1772) vs. **MyFlatMutableOffset::my_primitive** (p. 1772)

8.255.2 Member Function Documentation

8.255.2.1 set()

```
template<typename T >
bool  rti::flat::PrimitiveOffset< T >::set (
            T value )  [inline]
```

Sets a value for this primitive member.

Precondition

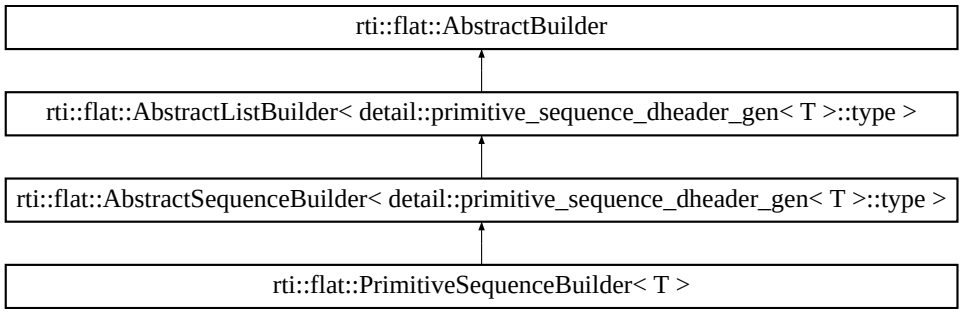
!is_null()

8.256 rti::flat::PrimitiveSequenceBuilder< T > Class Template Reference

Builds a sequence of primitive members.

`#include <SequenceBuilders.hpp>`

Inheritance diagram for rti::flat::PrimitiveSequenceBuilder< T >:



Public Member Functions

- **PrimitiveSequenceBuilder** & **add_next** (T value)

  Adds the next element.
- **PrimitiveSequenceBuilder** & **add_n** (const T *array, unsigned int count)

  Adds all the elements in an array.
- **PrimitiveSequenceBuilder** & **add_n** (unsigned int count, T value)

  Adds a number of elements with the same value.
- **PrimitiveSequenceBuilder** & **add_n** (unsigned int count)

  Adds a number of uninitialized elements.
- **Offset finish** ()

  Finishes building the sequence.

Additional Inherited Members

8.256.1 Detailed Description

template<typename T>
class rti::flat::PrimitiveSequenceBuilder< T >

Builds a sequence of primitive members.

The elements can be added one by one with add_next() or all at once with **add_n()** (p. 1956).

8.256.2 Member Function Documentation

8.256.2.1 add_next()

```
template<typename T >
PrimitiveSequenceBuilder & rti::flat::PrimitiveSequenceBuilder< T >::add_next (
            T value ) [inline]
```

Adds the next element.

Parameters

value	The primitive element to add

8.256.2.2 add_n() [1/3]

```
template<typename T >
PrimitiveSequenceBuilder & rti::flat::PrimitiveSequenceBuilder< T >::add_n (
            const T * array,
            unsigned int count ) [inline]
```

Adds all the elements in an array.

Parameters

array	The array containing the values to add
count	The size of the array

Precondition

array must not be NULL

8.256.2.3 add_n() [2/3]

```
template<typename T >
PrimitiveSequenceBuilder & rti::flat::PrimitiveSequenceBuilder< T >::add_n (
            unsigned int count,
            T value ) [inline]
```

Adds a number of elements with the same value.

This overload initializes all the elements to a value.

Note

This operation is O(`count`). The other overloads, **add_n(unsigned int)** (p. 1957) and **add_n(const T*, unsigned int)** (p. 1956), are more efficient

Parameters

count	The number of elements to add
value	The value to set for each element

8.256.2.4 add_n() [3/3]

```
template<typename T >
PrimitiveSequenceBuilder & rti::flat::PrimitiveSequenceBuilder< T >::add_n (
            unsigned int count ) [inline]
```

Adds a number of uninitialized elements.

This operation is O(1), since it leaves the elements uninitialized. They can be initialized using the Offset returned by **finish()** (p. 1957) and **rti::flat::plain_cast()** (p. 417). For example:

```
MyFlatMutableBuilder builder = ...;
auto seq_builder = builder.build_my_primitive_seq();
seq_builder.add_n(1000);
auto seq_offset = seq_builder.finish();
int32_t *elements = rti::flat::plain_cast(seq_offset);
for (int i = 0; i < 1000; i++) {
    elements[i] = ...;
}
```

Parameters

count	The number of elements to add

8.256.2.5 finish()

```
template<typename T >
Offset rti::flat::PrimitiveSequenceBuilder< T >::finish ( ) [inline]
```

Finishes building the sequence.

Returns

An Offset to the member that has been built.

See also

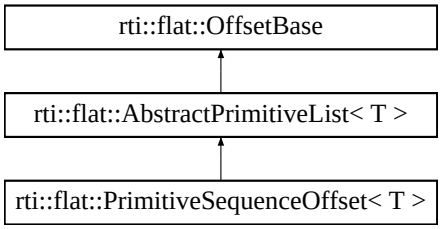
discard() (p. 784)

8.257 rti::flat::PrimitiveSequenceOffset< T > Class Template Reference

Offset to a sequence of primitive elements.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::PrimitiveSequenceOffset< T >:



Public Member Functions

- unsigned int **element_count** () const
  Returns the number of elements.

8.257.1 Detailed Description

template<typename T>
class rti::flat::PrimitiveSequenceOffset< T >

Offset to a sequence of primitive elements.

Template Parameters

T	The primitive type

8.257.2 Member Function Documentation

8.257.2.1 element_count()

```
template<typename T >
unsigned int  rti::flat::PrimitiveSequenceOffset< T >::element_count ( ) const  [inline]
```

Returns the number of elements.

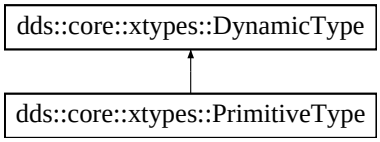
Referenced by **rti::flat::StringOffset::element_count()**.

8.258 dds::core::xtypes::PrimitiveType Class Reference

<<***value-type***>> *(p. 350)* Represents an IDL primitive type

`#include <dds/core/xtypes/PrimitiveTypes.hpp>`

Inheritance diagram for dds::core::xtypes::PrimitiveType:



Related Functions

(Note that these are not member functions.)

- template<typename T >
  const **PrimitiveType**< T > & **primitive_type** ()

  Obtains a singleton of PrimitiveType<T>

Additional Inherited Members

8.258.1 Detailed Description

<<***value-type***>> *(p. 350)* Represents an IDL primitive type

Template Parameters

T	The C++ primitive type for which to obtain a **DynamicType** (p. 1504). These are the possible types:

C++ type (T)	IDL type
char	char
bool	boolean
uint8_t	uint8
int8_t	int8
octet_tag_t	octet
(u)int16_t	(unsigned) short
(u)int32_t	(unsigned) long
(u)int64_t	(unsigned) long long
float	float
double	double
rti::core::LongDouble (p. 1696)	long double
wchar_t	wchar

Instead of instantiating this class it is recommended to obtain a singleton for each primitive type T using primitive_type.

Note: this class doesn't have any members other than those inherited from **DynamicType** (p. 1504).

See also

primitive_type() (p. 1960)

8.258.2 Friends And Related Function Documentation

8.258.2.1 primitive_type()

```
template<typename T >
const  PrimitiveType< T > & primitive_type ( )  [related]
```

Obtains a singleton of PrimitiveType<T>

8.259 rti::config::PrintFormat_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) PrintFormat.

```
#include <rti/config/Logger.hpp>
```

Public Types

- enum **type** { }
  The underlying `enum` *type.*

8.259.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) PrintFormat.

8.259.2 Member Enumeration Documentation

8.259.2.1 type

```
enum  rti::config::PrintFormat_def::type
```

The underlying `enum` type.

Enumerator

DEFAULT	(default) Print message, method name, log level, **Activity Context** (p. 449) (what was happening when the event occurred), and logging category.
TIMESTAMPED	Print message, method name, log level, **Activity Context** (p. 449), logging category, and timestamp.
VERBOSE	Print message with all available context information (includes thread identifier, message location).
VERBOSE_TIMESTAMPED	Print message with all available context information, and timestamp.
DEBUG	Print a set of fields (including message number and backtrace information) that may be useful for internal debugging.
MINIMAL	Print only message number and message location.
MAXIMAL	Print all available fields.

8.260 rti::topic::PrintFormatKind_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) PrintFormatKind.

`#include <PrintFormat.hpp>`

Public Types

- enum **type** {
  DEFAULT = DDS_DEFAULT_PRINT_FORMAT ,
  XML = DDS_XML_PRINT_FORMAT ,
  JSON = DDS_JSON_PRINT_FORMAT }

  The underlying `enum` type.

8.260.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) PrintFormatKind.

8.260.2 Member Enumeration Documentation

8.260.2.1 type

`enum` **rti::topic::PrintFormatKind_def::type**

The underlying `enum` type.

Enumerator

DEFAULT	Use a default format specific to RTI Connext to represent the data when converting a data sample to a string.
XML	Use an XML format to represent the data when converting a data sample to a string.
JSON	Use an JSON format to represent the data when converting a data sample to a string.

8.261 rti::topic::PrintFormatProperty Class Reference

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* A collection of attributes used to configure how data samples will be formatted when converted to a string.

`#include <rti/topic/PrintFormat.hpp>`

Public Member Functions

- **PrintFormatProperty** (**PrintFormatKind** the_kind=PrintFormatKind::DEFAULT, bool is_pretty_print=true, bool is_enum_as_int=false, bool is_include_root_elements=true)

  Initializes the properties.
- **PrintFormatKind kind** () const

  Get the value of kind.
- **PrintFormatProperty** & **kind** (**PrintFormatKind** value)

  Set the value for kind, indicating the kind of format to be used when converting a data sample to a string.
- bool **pretty_print** () const

  Get the value of pretty_print.
- **PrintFormatProperty** & **pretty_print** (bool value)

  Set the value for pretty_print, indicating if the string representation of the data sample should be a more readable format or should be void of all white space.
- bool **enum_as_int** () const

  Get the value of enum_as_int.
- **PrintFormatProperty** & **enum_as_int** (bool value)

  Set the value for enum_as_int, indicating whether to format enum values as integers or as strings when converting data samples to strings.
- bool **include_root_elements** () const

  Get the value of include_root_elements.
- **PrintFormatProperty** & **include_root_elements** (bool value)

  Set the value for include_root_elements, indicating whether to include the root elements of the print format in the output string when converting data samples to strings.

Static Public Member Functions

- static **PrintFormatProperty Default** ()

  Creates a property with PrintFormatKind::DEFAULT.
- static **PrintFormatProperty Xml** ()

  Creates a property with PrintFormatKind::XML.
- static **PrintFormatProperty Json** ()

  Creates a property with PrintFormatKind::JSON.

8.261.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* A collection of attributes used to configure how data samples will be formatted when converted to a string.

See also

rti::topic::to_string (p. 445)

8.261.2 Constructor & Destructor Documentation

8.261.2.1 PrintFormatProperty()

```
rti::topic::PrintFormatProperty::PrintFormatProperty (
              PrintFormatKind the_kind = PrintFormatKind::DEFAULT,
             bool is_pretty_print = true,
             bool is_enum_as_int = false,
             bool is_include_root_elements = true )  [inline]
```

Initializes the properties.

8.261.3 Member Function Documentation

8.261.3.1 kind() [1/2]

```
PrintFormatKind rti::topic::PrintFormatProperty::kind ( ) const  [inline]
```

Get the value of kind.

See also

kind(PrintFormatKind value) (p. 1963)

8.261.3.2 kind() [2/2]

```
PrintFormatProperty & rti::topic::PrintFormatProperty::kind (
             PrintFormatKind value )  [inline]
```

Set the value for kind, indicating the kind of format to be used when converting a data sample to a string.

Data samples can be represented in a default, XML, or JSON format.

See also

PrintFormatKind (p. 445)

[default] PrintFormatKind::DEFAULT_PRINT_FORMAT

Parameters

value	The value to set for kind

8.261.3.3 pretty_print() [1/2]

```
bool rti::topic::PrintFormatProperty::pretty_print ( ) const  [inline]
```

Get the value of pretty_print.

See also

pretty_print(bool value) (p. 1964)

8.261.3.4 pretty_print() [2/2]

```
PrintFormatProperty & rti::topic::PrintFormatProperty::pretty_print (
            bool value )  [inline]
```

Set the value for pretty_print, indicating if the string representation of the data sample should be a more readable format or should be void of all white space.

A value of true will add indentation and newlines to the string representation of the data sample making it more readable. For example:

```
<FooType>
   <myLong>5</myLong>
</FooType>
```

A value of false will cause all white space in the string representation of the data sample to be eliminated. For example:

```
<FooType><myLong>5</myLong></FooType>
```

[default] true

Parameters

value	The value to set for pretty_print

8.261.3.5 enum_as_int() [1/2]

```
bool rti::topic::PrintFormatProperty::enum_as_int ( ) const  [inline]
```

Get the value of enum_as_int.

See also

enum_as_int(bool value) (p. 1965)

8.261.3.6 enum_as_int() [2/2]

```
PrintFormatProperty & rti::topic::PrintFormatProperty::enum_as_int (
            bool value )  [inline]
```

Set the value for enum_as_int, indicating whether to format enum values as integers or as strings when converting data samples to strings.

A value of true will cause enumeration literals to be printed using their integer value. For example:

```
<FooType>
   <myEnum>5</myEnum>
</FooType>
```

A value of false will cause enumeration literals to be printed as strings using their member names. For example:

```
<FooType>
   <myEnum>RED</myEnum>
</FooType>
```

[default] false

Parameters

value	The value to set for enum_as_int

8.261.3.7 include_root_elements() [1/2]

```
bool rti::topic::PrintFormatProperty::include_root_elements ( ) const  [inline]
```

Get the value of include_root_elements.

See also

include_root_elements(bool value) (p. 1965)

8.261.3.8 include_root_elements() [2/2]

```
PrintFormatProperty & rti::topic::PrintFormatProperty::include_root_elements (
            bool value )  [inline]
```

Set the value for include_root_elements, indicating whether to include the root elements of the print format in the output string when converting data samples to strings.

This field only applies if the **PrintFormatProperty::kind** (p. 1963) is PrintFormatKind::XML_PRINT_FORMAT or PrintFormatKind::JSON_PRINT_FORMAT

If the value is true and the kind is PrintFormatKind::XML_PRINT_FORMAT then a top-level tag with the type's name in it will be included in the output. For example:

```
<FooType>
   <myLong>5</myLong>
</FooType>
```

If the value is true and the kind is PrintFormatKind::JSON_PRINT_FORMAT then top-level opening and closing braces will be included in the output. For example:

```
{
   "myLong":5
}
```

If the value is false the elements described above will not be included in the output. For example:

```
<myLong>5</myLong>
"myLong":5
```

[default] true

Parameters

value	The value to set for include_root_elements

8.261.3.9 Default()

```
static  PrintFormatProperty rti::topic::PrintFormatProperty::Default ( )  [inline], [static]
```

Creates a property with PrintFormatKind::DEFAULT.

8.261.3.10 Xml()

```
static  PrintFormatProperty rti::topic::PrintFormatProperty::Xml ( )  [inline], [static]
```

Creates a property with PrintFormatKind::XML.

8.261.3.11 Json()

```
static PrintFormatProperty rti::topic::PrintFormatProperty::Json ( ) [inline], [static]
```

Creates a property with PrintFormatKind::JSON.

8.262 rti::core::ProductVersion Class Reference

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Represents the current version of RTI Connext

```
#include <rti/core/ProductVersion.hpp>
```

Public Member Functions

- **ProductVersion** ()

  *Creates the **unknown()** (*p. *1969) product version.*
- std::string **to_string** () const

  Get the product version as a string.
- uint8_t **major_version** () const

  Get the major product version.
- uint8_t **minor_version** () const

  Get the minor product version.
- uint8_t **release_version** () const

  Get the release letter for product version.
- uint8_t **revision_version** () const

  Get the revision number of product.

Static Public Member Functions

- static **ProductVersion** **current** ()

  Returns the current version of RTI Connext.
- static **ProductVersion** **unknown** ()

  Returns the value to indicate that the version is unknown.

8.262.1 Detailed Description

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Represents the current version of RTI Connext

8.262.2 Constructor & Destructor Documentation

8.262.2.1 ProductVersion()

```
rti::core::ProductVersion::ProductVersion ( )  [inline]
```

Creates the **unknown()** (p. 1969) product version.

8.262.3 Member Function Documentation

8.262.3.1 to_string()

```
std::string rti::core::ProductVersion::to_string ( ) const  [inline]
```

Get the product version as a string.

8.262.3.2 major_version()

```
uint8_t rti::core::ProductVersion::major_version ( ) const  [inline]
```

Get the major product version.

8.262.3.3 minor_version()

```
uint8_t rti::core::ProductVersion::minor_version ( ) const  [inline]
```

Get the minor product version.

8.262.3.4 release_version()

```
uint8_t rti::core::ProductVersion::release_version ( ) const  [inline]
```

Get the release letter for product version.

8.262.3.5 revision_version()

```
uint8_t rti::core::ProductVersion::revision_version ( ) const  [inline]
```

Get the revision number of product.

8.262.3.6 current()

```
static  ProductVersion rti::core::ProductVersion::current ( )  [inline], [static]
```

Returns the current version of RTI Connext.

8.262.3.7 unknown()

```
static  ProductVersion rti::core::ProductVersion::unknown ( )  [inline], [static]
```

Returns the value to indicate that the version is unknown.

8.263 rti::core::policy::Property Class Reference

<<***extension***>> *(*p. *116)* Stores key/value string pairs that can configure certain parameters of RTI Connext not exposed through QoS policies. It can also store and propagate through the discovery mechanism application-specific information associated to a **dds::core::Entity** (p. 1520).

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Types

- typedef std::pair< std::string, std::string > **Entry**

  *The type of the elements that **Property** (*p. *1969) contains.*

Public Member Functions

- **Property** ()

  *Creates an empty **Property** (p. 1969) container.*
- template<typename EntryIter >
  Property (EntryIter begin, EntryIter end, bool is_propagate=false)

  *Creates a **Property** (p. 1969) container with the entries specified by an iterator range.*
- **Property** (std::initializer_list< **Entry** > entries)

  *<<**C++11**>> (p. 353) Creates a **Property** (p. 1969) container with entries from an initializer_list*
- bool **exists** (const std::string &key) const

  Returns true if a property exists.
- std::string **get** (const std::string &key) const

  Returns the value of a property identified by a key if it exists.
- **rti::core::optional_value**< std::string > **try_get** (const std::string &key) const

  *Returns the value of a property identified by a key or an empty **optional_value** (p. 1892) if it doesn't exist.*
- **Property** & **set** (const **Entry** &property, bool **propagate**=false)

  Adds or assigns a property from a pair of strings.
- template<typename EntryIter >
  Property & **set** (EntryIter begin, EntryIter end, bool is_propagate=false)

  Adds a range of properties.
- bool **remove** (const std::string &key)

  Removes the property identified by a key.
- size_t **size** () const

  Returns the number of properties.
- std::map< std::string, std::string > **get_all** () const

  Retrieves a copy of all the entries in a std::map.
- void **set** (std::initializer_list< **Entry** > entries, bool is_propagate=false)

  *<<**C++11**>> (p. 353) Adds properties from an initializer_list*
- bool **propagate** (const std::string &key) const

  Returns whether the 'propagate' attribute for a property is set or not.

8.263.1 Detailed Description

*<<**extension**>> (p. 116)* Stores key/value string pairs that can configure certain parameters of RTI Connext not exposed through QoS policies. It can also store and propagate through the discovery mechanism application-specific information associated to a **dds::core::Entity** (p. 1520).

Entity:

dds::domain::DomainParticipant (p. 1326) **dds::sub::DataReader** (p. 976) **dds::pub::DataWriter** (p. 1138)

Properties:

RxO (p. 514) = N/A;
Changeable (p. 514) = **YES** (p. 514)

See also

dds::sub::builtin_subscriber (p. 671)

8.263.2 Usage

The PROPERTY QoS policy can be used to associate a set of properties in the form of (name, value) pairs with a **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138), or **dds::domain::DomainParticipant** (p. 1326). This is similar to the **dds::core::policy::UserData** (p. 2591), except this policy uses (name, value) pairs, and you can select whether or not a particular pair should be propagated (included in the builtin topic).

You can find a complete list of predefined properties in the `Property Reference Guide`.

This QoS policy may be used to configure:

- Durable Writer History, see **Configuring Durable Writer History** (p. 101)
- Durable Reader State, see **Configuring Durable Reader State** (p. 102)
- Builtin Transport Plugins, see **UDPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 388), **UDPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 401), and **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 383)
- Extension Transport Plugins, see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 81)
- **Clock Selection** (p. 43)

In addition, you may add your own name/value pairs to the **Property** (p. 1969) QoS policy of an Entity. Via this QoS policy, you can direct RTI Connext to propagate these name/value pairs with the discovery information for the Entity. Applications that discover the Entity can then access the user-specific name/value pairs in the discovery information of the remote Entity. This allows you to add meta-information about an Entity for application-specific use, for example, authentication/authorization certificates (which can also be done using the **dds::core::policy::UserData** (p. 2591) or **dds::core::policy::GroupData** (p. 1591)).

8.263.2.1 Reasons for Using the PropertyQosPolicy

- Supports dynamic loading of extension transports
- Supports multiple instances of the builtin transports
- Allows full pluggable transport configuration for non-C/C++ language bindings (Java, .NET, etc.)
- Avoids the process of creating entities disabled, changing their QoS settings, then enabling them
- Allows selection of clock

Some of the RTI Connext capabilities configurable via the **Property** (p. 1969) QoS policy can also be configured in code via APIs. However, the **Property** (p. 1969) QoS policy allows you to configure those parameters via XML files. In addition, some of the configuration APIs will only work if the Entity was created in a disabled state and then enabled after the configuration change was applied. By configuring those parameters using the **Property** (p. 1969) QoS policy during entity creation, you avoid the additional work of first creating a disabled entity and then enabling it afterwards.

There are helper functions to facilitate working with properties, see the **PROPERTY** (p. 541) page.

8.263.3 Member Typedef Documentation

8.263.3.1 Entry

```
typedef std::pair<std::string, std::string>  rti::core::policy::Property::Entry
```

The type of the elements that **Property** (p. 1969) contains.

Key/value string pairs.

8.263.4 Constructor & Destructor Documentation

8.263.4.1 Property() [1/3]

```
rti::core::policy::Property::Property ( )  [inline]
```

Creates an empty **Property** (p. 1969) container.

8.263.4.2 Property() [2/3]

```
template<typename EntryIter >
rti::core::policy::Property::Property (
            EntryIter begin,
            EntryIter end,
            bool is_propagate = false )  [inline]
```

Creates a **Property** (p. 1969) container with the entries specified by an iterator range.

See also

set(EntryIter, EntryIter, bool) (p. 1974)

8.263.4.3 Property() [3/3]

```
rti::core::policy::Property::Property (
            std::initializer_list<  Entry > entries )  [inline]
```

<<***C++11***>> *(p. 353)* Creates a **Property** (p. 1969) container with entries from an initializer_list

For example:
```
Property my_properties {{"key 1", "value 1"}, {"key 2", "value 2"}};
```

If a key is duplicated, only one entry will be inserted, with the value that comes last.

8.263.5 Member Function Documentation

8.263.5.1 exists()

```
bool rti::core::policy::Property::exists (
            const std::string & key ) const  [inline]
```

Returns true if a property exists.

8.263.5.2 get()

```
std::string rti::core::policy::Property::get (
            const std::string & key ) const
```

Returns the value of a property identified by a key if it exists.

If the property doesn't exist it throws **dds::core::PreconditionNotMetError** (p. 1941).

8.263.5.3 try_get()

```
rti::core::optional_value< std::string > rti::core::policy::Property::try_get (
            const std::string & key ) const
```

Returns the value of a property identified by a key or an empty **optional_value** (p. 1892) if it doesn't exist.

8.263.5.4 set() [1/3]

```
Property & rti::core::policy::Property::set (
            const  Entry & property,
            bool propagate = false )
```

Adds or assigns a property from a pair of strings.

If the key doesn't exists it adds the new entry, otherwise it overrides its value with the new one.

Parameters

property	`property.first` is the key and `property.second` is the value
propagate	Indicates if the property must be propagated on discovery

8.263.5.5 set() [2/3]

```
template<typename EntryIter >
Property & rti::core::policy::Property::set (
            EntryIter begin,
            EntryIter end,
            bool is_propagate = false )  [inline]
```

Adds a range of properties.

If a key is duplicated, only one entry will remain, with the value that comes last.

Template Parameters

EntryIter	A forward iterator whose value type is **Property::Entry** (p. 1972), such as the iterators of a `std::map<std::string, std::string>`.

Parameters

begin	Beginning of a range of Entry
end	End of that range
is_propagate	Indicates if the property must be propagated on discovery

8.263.5.6 remove()

```
bool rti::core::policy::Property::remove (
            const std::string & key )
```

Removes the property identified by a key.

Returns

true if the property was removed or false if it didn't exist.

8.263.5.7 size()

```
size_t rti::core::policy::Property::size ( ) const
```

Returns the number of properties.

8.263.5.8 get_all()

```
std::map< std::string, std::string > rti::core::policy::Property::get_all ( ) const
```

Retrieves a copy of all the entries in a std::map.

8.263.5.9 set() [3/3]

```
void rti::core::policy::Property::set (
            std::initializer_list<  Entry > entries,
            bool is_propagate = false )  [inline]
```

<<***C++11***>> *(*p. *353)* Adds properties from an initializer_list

8.263.5.10 propagate()

```
bool rti::core::policy::Property::propagate (
            const std::string & key ) const
```

Returns whether the 'propagate' attribute for a property is set or not.

8.264 rti::core::ProtocolVersion Class Reference

<<***extension***>> *(*p. *354)* Represents the current version of RTI Connext

```
#include <rti/core/ProtocolVersion.hpp>
```

Public Member Functions

- **ProtocolVersion** ()

  Creates an invalid protocol version.
- **ProtocolVersion** (uint8_t major, uint8_t minor)

  Creates a version with the given major and minor values.
- uint8_t **major_version** () const

  The major version number.
- uint8_t **minor_version** () const

  The minor version number.

Static Public Member Functions

- static **ProtocolVersion current** ()

  The most recent protocol version. Currently 2.1.

8.264.1 Detailed Description

<<***extension***>> *(*p. *354)* Represents the current version of RTI Connext

8.264.2 Constructor & Destructor Documentation

8.264.2.1 ProtocolVersion() [1/2]

```
rti::core::ProtocolVersion::ProtocolVersion ( )  [inline]
```

Creates an invalid protocol version.

8.264.2.2 ProtocolVersion() [2/2]

```
rti::core::ProtocolVersion::ProtocolVersion (
            uint8_t major,
            uint8_t minor )  [inline]
```

Creates a version with the given major and minor values.

8.264.3 Member Function Documentation

8.264.3.1 major_version()

```
uint8_t rti::core::ProtocolVersion::major_version ( ) const  [inline]
```

The major version number.

8.264.3.2 minor_version()

```
uint8_t rti::core::ProtocolVersion::minor_version ( ) const  [inline]
```

The minor version number.

8.264.3.3 current()

```
static  ProtocolVersion rti::core::ProtocolVersion::current ( )  [inline], [static]
```

The most recent protocol version. Currently 2.1.

8.265 dds::topic::PublicationBuiltinTopicData Class Reference

Entry created when a **dds::pub::DataWriter** (p. 1138) is discovered in association with its **dds::pub::Publisher** (p. 1993).

```
#include <dds/topic/BuiltinTopic.hpp>
```

Public Member Functions

- **TPublicationBuiltinTopicData** ()

  *Create a default **PublicationBuiltinTopicData** (p. 1977).*
- const **dds::topic::BuiltinTopicKey** & **key** () const

  Get the DCPS key to distinguish entries.
- const **dds::topic::BuiltinTopicKey** & **participant_key** () const

  *Get the DCPS key of the **dds::domain::DomainParticipant** (p. 1326) to which the **dds::pub::DataWriter** (p. 1138) belongs.*
- const **dds::core::string** & **topic_name** () const

  *Get the name of the related **dds::topic::Topic** (p. 2474).*
- const **dds::core::string** & **type_name** () const

  *Get the name of the type attached to the **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::Durability** & **durability** () const

  *Get the **dds::core::policy::Durability** (p. 1433) policy of the corresponding **dds::pub::DataWriter** (p. 1138).*
- const **dds::core::policy::DurabilityService** & **durability_service** () const

*Get the **dds::core::policy::DurabilityService*** (p. *1443*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).

- const **dds::core::policy::Deadline** & **deadline** () const

  *Get the **dds::core::policy::Deadline*** (p. *1255*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::LatencyBudget** & **latency_budget** () const

  *Get the **dds::core::policy::LatencyBudget*** (p. *1631*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::Liveliness** & **liveliness** () const

  *Get the **dds::core::policy::Liveliness*** (p. *1645*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::Reliability** & **reliability** () const

  *Get the **dds::core::policy::Reliability*** (p. *2149*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::Lifespan** & **lifespan** () const

  *Get the **dds::core::policy::Lifespan*** (p. *1635*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::UserData** & **user_data** () const

  *Get the **dds::core::policy::UserData*** (p. *2591*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::Ownership** & **ownership** () const

  *Get the **dds::core::policy::Ownership*** (p. *1899*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::OwnershipStrength** & **ownership_strength** () const

  *Get the **dds::core::policy::OwnershipStrength*** (p. *1906*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::DestinationOrder** & **destination_order** () const

  *Get the **dds::core::policy::DestinationOrder*** (p. *1258*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::Presentation** & **presentation** () const

  *Get the **dds::core::policy::Presentation*** (p. *1942*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::Partition** & **partition** () const

  *Get the **dds::core::policy::Partition*** (p. *1923*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::TopicData** & **topic_data** () const

  *Get the **dds::core::policy::TopicData*** (p. *2500*) *policy of the related **dds::topic::Topic*** (p. *2474*).
- const **dds::core::policy::GroupData** & **group_data** () const

  *Get the **dds::core::policy::GroupData*** (p. *1591*) *policy of the **dds::pub::Publisher*** (p. *1993*) *to which the **dds::pub::DataWriter*** (p. *1138*) *belongs.*
- const **dds::core::policy::DataRepresentation** & **representation** () const

  *Get the **dds::core::policy::DataRepresentation*** (p. *1113*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::DataTag** & **data_tag** () const

  *Get the **dds::core::policy::DataTag*** (p. *1132*) *policy of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::policy::DataRepresentation** & **representation** () const

  Get the publication data representation.
- **dds::core::optional**< **dds::core::xtypes::DynamicType** > **type** () const

  <<***extension***>> (p. *354*) *Get the type*
- const **dds::core::optional**< **dds::core::xtypes::DynamicType** > & **get_type_no_copy** () const

  <<***extension***>> (p. *354*) *Get the type by reference if possible*
- const **dds::topic::BuiltinTopicKey** & **publisher_key** () const

  <<***extension***>> (p. *354*) *Get the DCPS key of the **dds::pub::Publisher*** (p. *1993*) *to which the **dds::pub::DataWriter*** (p. *1138*) *belongs*
- const **rti::core::policy::Property** & **property** () const

  <<***extension***>> (p. *354*) *Get the propagated name-value properties of the corresponding **dds::pub::DataWriter*** (p. *1138*).
- const **dds::core::vector**< **rti::core::Locator** > & **unicast_locators** () const

  <<***extension***>> (p. *354*) *Get the custom unicast locators that the endpoint can specify.*
- const **rti::core::Guid** & **virtual_guid** () const

  <<***extension***>> (p. *354*) *Get the virtual **rti::core::Guid*** (p. *1596*) *associated to the **dds::pub::DataWriter*** (p. *1138*).

- const **rti::core::ProtocolVersion** & **rtps_protocol_version** () const

  <<***extension***>> *(p. 354) Get the version number of the RTPS wire protocol used.*
- const **rti::core::VendorId** & **rtps_vendor_id** () const

  <<***extension***>> *(p. 354) Get the ID of the vendor implementing the RTPS wire protocol.*
- const **rti::core::ProductVersion** & **product_version** () const

  <<***extension***>> *(p. 354) Get the current version for RTI Connext.*
- const **rti::core::policy::LocatorFilter** & **locator_filter** () const

  <<***extension***>> *(p. 354) Get the locator filters of the corresponding **dds::pub::DataWriter** (p. 1138)*
- bool **disable_positive_acks** () const

  <<***extension***>> *(p. 354) See whether or not a matching **dds::sub::DataReader** (p. 976) will send positive acknowledgments for reliability.*
- const **rti::core::policy::EntityName** & **publication_name** () const

  <<***extension***>> *(p. 354) Get the publication name and role name.*
- const **rti::topic::trust::EndpointTrustProtectionInfo** & **trust_protection_info** () const

  <<***extension***>> *(p. 354) Get the Trust Plugins information associated with the discovered DataWriter.*
- const **rti::topic::trust::EndpointTrustAlgorithmInfo** & **trust_algorithm_info** () const

  <<***extension***>> *(p. 354) Get the Trust plugins algorithms associated with the discovered DataWriter.*
- const **dds::topic::BuiltinTopicKey** & **related_datareader_key** () const

  *Get the **dds::topic::BuiltinTopicKey** (p. 905) of the related **dds::sub::DataReader** (p. 976) of this DataWriter.*
- const **rti::core::policy::Service** & **service** () const

  <<***extension***>> *(p. 354) Get the Service policy*

8.265.1 Detailed Description

Entry created when a **dds::pub::DataWriter** (p. 1138) is discovered in association with its **dds::pub::Publisher** (p. 1993).

Data associated with the built-in topic **dds::topic::publication_topic_name()** (p. 443). It contains QoS policies and additional information that apply to the remote **dds::pub::DataWriter** (p. 1138) the related **dds::pub::Publisher** (p. 1993).

8.265.2 Member Function Documentation

8.265.2.1 TPublicationBuiltinTopicData()

```
dds::topic::PublicationBuiltinTopicData::TPublicationBuiltinTopicData ( )  [inline]
```

Create a default **PublicationBuiltinTopicData** (p. 1977).

8.265.2.2 key()

```
const dds::topic::BuiltinTopicKey & dds::topic::PublicationBuiltinTopicData::key ( ) const [inline]
```

Get the DCPS key to distinguish entries.

Returns

The key

8.265.2.3 participant_key()

```
const dds::topic::BuiltinTopicKey & dds::topic::PublicationBuiltinTopicData::participant_key ( )
const [inline]
```

Get the DCPS key of the **dds::domain::DomainParticipant** (p. 1326) to which the **dds::pub::DataWriter** (p. 1138) belongs.

Returns

The DomainParticipant's key

8.265.2.4 topic_name()

```
const dds::core::string & dds::topic::PublicationBuiltinTopicData::topic_name ( ) const [inline]
```

Get the name of the related **dds::topic::Topic** (p. 2474).

Returns

The topic name

8.265.2.5 type_name()

```
const dds::core::string & dds::topic::PublicationBuiltinTopicData::type_name ( ) const [inline]
```

Get the name of the type attached to the **dds::topic::Topic** (p. 2474).

Returns

The type name

8.265.2.6 durability()

```
const dds::core::policy::Durability & dds::topic::PublicationBuiltinTopicData::durability ( )
const [inline]
```

Get the **dds::core::policy::Durability** (p. 1433) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The Durability policy

8.265.2.7 durability_service()

```
const dds::core::policy::DurabilityService & dds::topic::PublicationBuiltinTopicData::durability↩
_service ( ) const [inline]
```

Get the **dds::core::policy::DurabilityService** (p. 1443) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The DurabilityService policy

8.265.2.8 deadline()

```
const dds::core::policy::Deadline & dds::topic::PublicationBuiltinTopicData::deadline ( ) const
[inline]
```

Get the **dds::core::policy::Deadline** (p. 1255) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The Deadline policy

8.265.2.9 latency_budget()

```
const dds::core::policy::LatencyBudget & dds::topic::PublicationBuiltinTopicData::latency_budget
( ) const [inline]
```

Get the **dds::core::policy::LatencyBudget** (p. 1631) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The LatencyBudget policy

8.265.2.10 liveliness()

```
const dds::core::policy::Liveliness & dds::topic::PublicationBuiltinTopicData::liveliness ( )
const [inline]
```

Get the **dds::core::policy::Liveliness** (p. 1645) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The Liveliness policy

8.265.2.11 reliability()

```
const dds::core::policy::Reliability & dds::topic::PublicationBuiltinTopicData::reliability ( )
const [inline]
```

Get the **dds::core::policy::Reliability** (p. 2149) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The Reliability policy

8.265.2.12 lifespan()

```
const dds::core::policy::Lifespan & dds::topic::PublicationBuiltinTopicData::lifespan ( ) const
[inline]
```

Get the **dds::core::policy::Lifespan** (p. 1635) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The Lifespan policy

8.265.2.13 user_data()

```
const dds::core::policy::UserData & dds::topic::PublicationBuiltinTopicData::user_data ( ) const
[inline]
```

Get the **dds::core::policy::UserData** (p. 2591) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The UserData policy

8.265.2.14 ownership()

```
const dds::core::policy::Ownership & dds::topic::PublicationBuiltinTopicData::ownership ( ) const
[inline]
```

Get the **dds::core::policy::Ownership** (p. 1899) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The Ownership policy

8.265.2.15 ownership_strength()

```
const dds::core::policy::OwnershipStrength & dds::topic::PublicationBuiltinTopicData::ownership↩
_strength ( ) const [inline]
```

Get the **dds::core::policy::OwnershipStrength** (p. 1906) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The OwnershipStrength policy

8.265.2.16 destination_order()

```
const dds::core::policy::DestinationOrder & dds::topic::PublicationBuiltinTopicData::destination↩
_order ( ) const [inline]
```

Get the **dds::core::policy::DestinationOrder** (p. 1258) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The DestinationOrder policy

8.265.2.17 presentation()

```
const dds::core::policy::Presentation & dds::topic::PublicationBuiltinTopicData::presentation ( )
const [inline]
```

Get the **dds::core::policy::Presentation** (p. 1942) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The Presentation policy

8.265.2.18 partition()

```
const dds::core::policy::Partition & dds::topic::PublicationBuiltinTopicData::partition ( ) const
[inline]
```

Get the **dds::core::policy::Partition** (p. 1923) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The Partition policy

References **dds::core::Value**< **D** >**::delegate()**.

8.265.2.19 topic_data()

```
const dds::core::policy::TopicData & dds::topic::PublicationBuiltinTopicData::topic_data ( )
const [inline]
```

Get the **dds::core::policy::TopicData** (p. 2500) policy of the related **dds::topic::Topic** (p. 2474).

Returns

The TopicData policy

References **dds::core::Value**< **D** >**::delegate()**.

8.265.2.20 group_data()

```
const dds::core::policy::GroupData & dds::topic::PublicationBuiltinTopicData::group_data ( )
const [inline]
```

Get the **dds::core::policy::GroupData** (p. 1591) policy of the **dds::pub::Publisher** (p. 1993) to which the **dds::pub**↩ **::DataWriter** (p. 1138) belongs.

Returns

The GroupData policy

References **dds::core::Value**< **D** >**::delegate()**.

8.265.2.21 representation() [1/2]

```
const dds::core::policy::DataRepresentation & dds::topic::PublicationBuiltinTopicData::representation
( ) const [inline]
```

Get the **dds::core::policy::DataRepresentation** (p. 1113) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The DataRepresentation policy

References **dds::core::Value**< **D** >**::delegate()**.

8.265.2.22 data_tag()

```
const dds::core::policy::DataTag & dds::topic::PublicationBuiltinTopicData::data_tag ( ) const
[inline]
```

Get the **dds::core::policy::DataTag** (p. 1132) policy of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The DataTag policy

References **dds::core::Value**< **D** >**::delegate()**.

8.265.2.23 representation() [2/2]

```
const dds::core::policy::DataRepresentation & representation ( ) const
```

Get the publication data representation.

Returns

The **dds::core::policy::DataRepresentation** (p. 1113) policy of the corresponding DataWriter

Examples

Foo.hpp.

8.265.2.24 type()

```
dds::core::optional< dds::core::xtypes::DynamicType > type ( ) const
```

<<***extension***>> *(p. 354)* Get the type

Note

This is not a lightweight getter–obtaining the type may require some processing so it is recommended that you keep the value if you need to use it multiple times rather than look it up every time. See also **get_type_no_copy()** (p. 1986).

Returns

The type or an unset optional value if the type is not available

Examples

Foo.hpp.

8.265.2.25 get_type_no_copy()

```
const dds::core::optional< dds::core::xtypes::DynamicType > & get_type_no_copy ( ) const
```

<<***extension***>> *(p. 354)* Get the type by reference if possible

Exceptions

dds::core::PreconditionNotMet	If the type is available but is not in a format that is directly accessible. In that case it needs some extra processing and has to be accessed using **type()** (p. 1985). You can also ensure that it is in the right format by setting **rti::core::policy::↩ DomainParticipantResourceLimits::type_code_max_serialized_length()** (p. 1418) to 0.

Returns

The type or an unset optional value if the type is not available

See also

type() (p. 1985)

8.265.2.26 publisher_key()

```
const dds::topic::BuiltinTopicKey & publisher_key ( ) const
```

<<***extension***>> *(p. 354)* Get the DCPS key of the **dds::pub::Publisher** (p. 1993) to which the **dds::pub::DataWriter** (p. 1138) belongs

Returns

The key

8.265.2.27 property()

```
const rti::core::policy::Property & property ( ) const
```

<<***extension***>> *(p. 354)* Get the propagated name-value properties of the corresponding **dds::pub::DataWriter** (p. 1138).

Returns

The properties

8.265.2.28 unicast_locators()

```
const dds::core::vector< rti::core::Locator > & unicast_locators ( ) const
```

<<***extension***>> *(p. 354)* Get the custom unicast locators that the endpoint can specify.

The default locators will be used if this is not specified.

Returns

The custom unicast locators specified by the corresponding DataWriter

8.265.2.29 virtual_guid()

```
const rti::core::Guid & virtual_guid ( ) const
```

<<***extension***>> *(p. 354)* Get the virtual **rti::core::Guid** (p. 1596) associated to the **dds::pub::DataWriter** (p. 1138).

Returns

The virtual guid

8.265.2.30 rtps_protocol_version()

```
const rti::core::ProtocolVersion & rtps_protocol_version ( ) const
```

<<***extension***>> *(p. 354)* Get the version number of the RTPS wire protocol used.

Returns

The RTPS wire protocol version

8.265.2.31 rtps_vendor_id()

```
const rti::core::VendorId & rtps_vendor_id ( ) const
```

<<***extension***>> *(p. 354)* Get the ID of the vendor implementing the RTPS wire protocol.

Returns

The vendor ID

8.265.2.32 product_version()

```
const rti::core::ProductVersion & product_version ( ) const
```

<<***extension***>> *(p. 354)* Get the current version for RTI Connext.

Returns

The

8.265.2.33 locator_filter()

```
const rti::core::policy::LocatorFilter & locator_filter ( ) const
```

<<***extension***>> *(p. 354)* Get the locator filters of the corresponding **dds::pub::DataWriter** (p. 1138)

Returns

The rti::core::LocatorFilterQosPolicy

See also

rti::core::policy::MultiChannel (p. 1749)

8.265.2.34 disable_positive_acks()

```
bool disable_positive_acks ( ) const
```

<<***extension***>> *(p. 354)* See whether or not a matching **dds::sub::DataReader** (p. 976) will send positive acknowledgments for reliability.

Returns

true if positive acks are disabled, false otherwise

8.265.2.35 publication_name()

```
const  rti::core::policy::EntityName & publication_name ( ) const
```

<<***extension***>> *(p. 354)* Get the publication name and role name.

Returns

The **rti::core::policy::EntityName** (p. 1530) policy of the corresponding DataWriter

8.265.2.36 trust_protection_info()

```
const  rti::topic::trust::EndpointTrustProtectionInfo & trust_protection_info ( ) const
```

<<***extension***>> *(p. 354)* Get the Trust Plugins information associated with the discovered DataWriter.

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged data and metadata.

trust_protection_info contains information about how RTPS wire serialization, discovery, and liveliness interact with the loaded Trust Plugins. Two endpoints will not match if their trust_protection_info is incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.265.2.37 trust_algorithm_info()

```
const rti::topic::trust::EndpointTrustAlgorithmInfo & trust_algorithm_info ( ) const
```

<<***extension***>> *(p. 354)* Get the Trust plugins algorithms associated with the discovered DataWriter.

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged data and metadata.

trust_algorithm_info contains information about what algorithms the loaded Trust Plugins are running. Two endpoints will not match if their trust_algorithm_info are incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.265.2.38 related_datareader_key()

```
const dds::topic::BuiltinTopicKey & related_datareader_key ( ) const
```

Get the **dds::topic::BuiltinTopicKey** (p. 905) of the related **dds::sub::DataReader** (p. 976) of this DataWriter.

This field will be set to something other than **dds::topic::BuiltinTopicKey::unknown()** (p. 906) only when the discovered DataWriter is a part of Remote Procedure Call (RPC) or Request-Reply endpoints.

8.265.2.39 service()

```
const rti::core::policy::Service & service ( ) const
```

<<***extension***>> *(p. 354)* Get the Service policy

8.266 dds::core::status::PublicationMatchedStatus Class Reference

Information about the status **dds::core::status::StatusMask::publication_matched()** (p. 2384)

`#include <TStatus.hpp>`

Public Member Functions

- int32_t **total_count** () const

  *The total cumulative number of times that this **dds::pub::DataWriter** (p. 1138) discovered a "match" with a **dds::sub::DataReader** (p. 976).*
- int32_t **total_count_change** () const

  The changes in total_count since the last time the listener was called or the status was read.
- int32_t **current_count** () const

  *The current number of DataReaders with which this **dds::pub::DataWriter** (p. 1138) is matched.*
- int32_t **current_count_change** () const

  The change in current_count since the last time the listener was called or the status was read.
- const **dds::core::InstanceHandle last_subscription_handle** () const

  *This **InstanceHandle** (p. 1612) can be used to look up which remote **dds::sub::DataReader** (p. 976) was the last to cause this DataWriter's status to change, using **dds::pub::matched_subscription_data()** (p. 650).*
- int32_t **current_count_peak** () const

  <<***extension***>> *(p. 354) Greatest number of DataReaders that matched this **dds::pub::DataWriter** (p. 1138) simultaneously.*

8.266.1 Detailed Description

Information about the status **dds::core::status::StatusMask::publication_matched()** (p. 2384)

A "match" happens when the **dds::pub::DataWriter** (p. 1138) finds a **dds::sub::DataReader** (p. 976) with the same **dds::topic::Topic** (p. 2474), same or compatible data type, and requested QoS that is compatible with that offered by the **dds::pub::DataWriter** (p. 1138). (For information on compatible data types, see the `Extensible Types Guide`.)

This status is also changed (and the listener, if any, called) when a match is ended. A local **dds::pub::DataWriter** (p. 1138) will become "unmatched" from a remote **dds::sub::DataReader** (p. 976) when that **dds::sub::DataReader** (p. 976) goes away for any of the following reasons:

- The matched **dds::sub::DataReader** (p. 976)'s **dds::domain::DomainParticipant** (p. 1326) has lost liveliness.
- This DataWriter or the matched DataReader has changed QoS such that the entities are now incompatible.
- The matched DataReader has been deleted.

This status may reflect changes from multiple match or unmatch events, and the **dds::core::status::PublicationMatchedStatus::current_count_change** (p. 1992) can be used to determine the number of changes since the listener was called back or the status was checked.

8.266.2 Member Function Documentation

8.266.2.1 total_count()

```
int32_t dds::core::status::PublicationMatchedStatus::total_count ( ) const  [inline]
```

The total cumulative number of times that this **dds::pub::DataWriter** (p. 1138) discovered a "match" with a **dds::sub::DataReader** (p. 976).

This number increases whenever a new match is discovered. It does not decrease when an existing match goes away for any of the reasons described in **dds::core::status::PublicationMatchedStatus** (p. 1990).

8.266.2.2 total_count_change()

```
int32_t dds::core::status::PublicationMatchedStatus::total_count_change ( ) const  [inline]
```

The changes in total_count since the last time the listener was called or the status was read.

Note that this number will never be negative (because it's the total number of times this DataWriter ever matched with a DataReader).

8.266.2.3 current_count()

```
int32_t dds::core::status::PublicationMatchedStatus::current_count ( ) const  [inline]
```

The current number of DataReaders with which this **dds::pub::DataWriter** (p. 1138) is matched.

This number increases when a new match is discovered and decreases when an existing match goes away for any of the reasons described in **dds::core::status::PublicationMatchedStatus** (p. 1990).

8.266.2.4 current_count_change()

```
int32_t dds::core::status::PublicationMatchedStatus::current_count_change ( ) const  [inline]
```

The change in current_count since the last time the listener was called or the status was read.

Note that a negative current_count_change means that one or more DataReaders have become unmatched for one or more of the reasons described in **dds::core::status::PublicationMatchedStatus** (p. 1990).

8.266.2.5 last_subscription_handle()

```
const  dds::core::InstanceHandle dds::core::status::PublicationMatchedStatus::last_subscription_↩
handle ( ) const  [inline]
```

This **InstanceHandle** (p. 1612) can be used to look up which remote **dds::sub::DataReader** (p. 976) was the last to cause this DataWriter's status to change, using **dds::pub::matched_subscription_data()** (p. 650).

If the DataReader no longer matches this DataWriter due to any of the reasons in **dds::core::status::Publication↩ MatchedStatus** (p. 1990) except incompatible QoS, then the DataReader has been purged from this DataWriter's DomainParticipant discovery database. (See the "Discovery Overview" section of the User's Manual.) In that case, the **dds::pub::matched_subscription_data()** (p. 650) method will not be able to return information about the DataReader. The only way to get information about the lost DataReader is if you cached the information previously.

8.266.2.6 current_count_peak()

```
int32_t current_count_peak ( ) const
```

<<***extension***>> *(p. 354)* Greatest number of DataReaders that matched this **dds::pub::DataWriter** (p. 1138) simultaneously.

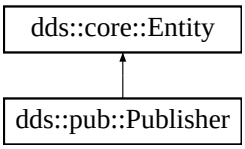
That is, there was no moment in time when more than this many DataReaders matched this DataWriter. (As a result, total_count can be higher than current_count_peak.)

8.267 dds::pub::Publisher Class Reference

<<***reference-type***>> *(p. 351)* A publisher is the object responsible for the actual dissemination of publications.

`#include <dds/pub/Publisher.hpp>`

Inheritance diagram for dds::pub::Publisher:



Public Member Functions

- **Publisher** (const **dds::domain::DomainParticipant** &the_participant)

  *Create a **Publisher** (p. 1993).*
- **Publisher** (const **dds::domain::DomainParticipant** &the_participant, const **dds::pub::qos::PublisherQos** &the_qos, **dds::pub::PublisherListener** *the_listener=NULL, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Create a **Publisher** (p. 1993) with the desired QoS policies and attaches to it the specified **PublisherListener** (p. 2006).*
- **Publisher** (const **::dds::domain::DomainParticipant** &the_participant, const **dds::pub::qos::PublisherQos** &the_qos, std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &mask= **dds**↩ **::core::status::StatusMask::all**())

  *Create a **Publisher** (p. 1993) with the desired QoS policies and attaches to it the specified **PublisherListener** (p. 2006).*
- const **dds::pub::qos::PublisherQos** **qos** () const

  Gets the PublisherQos.
- void **qos** (const **dds::pub::qos::PublisherQos** &the_qos)

  Set the PublisherQos.
- **Publisher** & **operator**<< (const **dds::pub::qos::PublisherQos** &the_qos)

  Sets the PublisherQos.
- **Publisher** & **operator**>> (**dds::pub::qos::PublisherQos** &the_qos)

  Gets the PublisherQos.
- **Publisher** & **default_datawriter_qos** (const **dds::pub::qos::DataWriterQos** &dwqos)

  Sets the default DataWriterQos.
- **dds::pub::qos::DataWriterQos** **default_datawriter_qos** () const

  Gets the default DataWriterQos.
- void **listener** (**Listener** *plistener, const **dds::core::status::StatusMask** &event_mask)

  *Sets the **Publisher** (p. 1993) listener.*
- **Listener** * **listener** () const

  *Returns the listener currently associated with this **Publisher** (p. 1993).*
- void **set_listener** (std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &event_↩ mask)

  Sets the listener associated with this publisher.
- void **set_listener** (std::shared_ptr< **Listener** > the_listener)

Sets the listener associated with this publisher.

- std::shared_ptr< **Listener** > **get_listener** () const

  Returns the listener currently associated with this ***Publisher*** *(*p. *1993).*

- void **wait_for_acknowledgments** (const **dds::core::Duration** &max_wait)

  Blocks the calling thread until all data written by this ***Publisher*** *(*p. *1993)'s reliable DataWriters is acknowledged, or until timeout expires.*

- const **dds::domain::DomainParticipant** & **participant** () const

  Gets the DomainParticipant to which this ***Publisher*** *(*p. *1993) belongs.*

- void **wait_for_asynchronous_publishing** (const **dds::core::Duration** &max_wait)

  *<<**extension**>> (*p. *354) Blocks the calling thread until asynchronous sending is complete.*

- void **close_contained_entities** ()

  *<<**extension**>> (*p. *354) Closes all the entities created from this* ***Publisher*** *(*p. *1993)*

Related Functions

(Note that these are not member functions.)

- template<typename PublisherForwardIterator >
  uint32_t **find_publishers** (const **dds::domain::DomainParticipant participant**, PublisherForwardIterator begin, uint32_t max_size)

  *<<**extension**>> (*p. *354) Retrieve all of the* ***dds::pub::Publisher*** *(*p. *1993) created from this* ***dds::domain::DomainParticipant*** *(*p. *1326)*

- template<typename PublisherBackInsertIterator >
  uint32_t **find_publishers** (const **dds::domain::DomainParticipant participant**, PublisherBackInsertIterator begin)

  Retrieve all the ***dds::pub::Publisher*** *(*p. *1993) created from this* ***dds::domain::DomainParticipant*** *(*p. *1326).*

- **dds::pub::Publisher find_publisher** (const **dds::domain::DomainParticipant participant**, const std::string &publisher_name)

  *<<**extension**>> (*p. *354) Looks up a* ***dds::pub::Publisher*** *(*p. *1993) by its entity name within the* ***dds::domain::DomainParticipant*** *(*p. *1326).*

- **dds::pub::Publisher implicit_publisher** (const **dds::domain::DomainParticipant** &dp)

  *<<**extension**>> (*p. *354) Get the implicit* ***dds::pub::Publisher*** *(*p. *1993) for a given* ***dds::domain::DomainParticipant*** *(*p. *1326).*

8.267.1 Detailed Description

*<<**reference-type**>> (*p. *351)* A publisher is the object responsible for the actual dissemination of publications.

QoS:

dds::pub::qos::PublisherQos (p. 2007)

Listener:

PublisherListener (p. 2006)

A publisher acts on the behalf of one or several **dds::pub::DataWriter** (p. 1138) objects that belong to it. When it is informed of a change to the data associated with one of its **dds::pub::DataWriter** (p. 1138) objects, it decides when it is appropriate to actually send the data-update message. In making this decision, it considers any extra information that goes with the data (timestamp, writer, etc.) as well as the QoS of the **dds::pub::Publisher** (p. 1993) and the **dds::pub::DataWriter** (p. 1138).

The following operations may be called even if the **dds::pub::Publisher** (p. 1993) is not enabled. Other operations will fail with the value **dds::core::NotEnabledError** (p. 1869) if called on a disabled **dds::pub::Publisher** (p. 1993):

- **dds::core::Entity::enable** (p. 1524),
- **dds::pub::Publisher::qos(const dds::pub::qos::PublisherQos&)** (p. 1997), **dds::pub::Publisher::qos() const** (p. 1997)
- **dds::pub::Publisher::set_listener** (p. 1999), **dds::pub::Publisher::get_listener** (p. 2000),
- **dds::core::cond::StatusCondition::StatusCondition(const dds::core::Entity&)** (p. 2373), **dds::core::Entity::status_changes()** (p. 1525)
- **dds::pub::DataWriter** (p. 1138) constructors, **dds::pub::DataWriter::close()** (p. 1525),
- **dds::pub::Publisher::default_datawriter_qos(const dds::pub::qos::DataWriterQos&)** (p. 1998), **dds::pub::Publisher::default_datawriter_qos()** (p. 1998), **dds::pub::Publisher::wait_for_acknowledgments** (p. 2001),

See also

Operations Allowed in Listener Callbacks (p. 1639)

dds::pub::find() (p. 651) and related functions

dds::pub::SuspendedPublication (p. 2441)

Publisher Use Cases (p. 310)

Entity Use Cases (p. 325)

Examples

Foo_publisher.cxx.

8.267.2 Constructor & Destructor Documentation

8.267.2.1 Publisher() [1/3]

```
dds::pub::Publisher::Publisher (
            const  dds::domain::DomainParticipant & the_participant )  [inline], [explicit]
```

Create a **Publisher** (p. 1993).

It uses default **qos::PublisherQos** (p. 2007) and no listener.

Parameters

the_participant	The DomainParticipant where this **Publisher** (p. 1993) will be created

8.267.2.2 Publisher() [2/3]

```
dds::pub::Publisher::Publisher (
            const  dds::domain::DomainParticipant & the_participant,
            const  dds::pub::qos::PublisherQos & the_qos,
             dds::pub::PublisherListener * the_listener = NULL,
            const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a **Publisher** (p. 1993) with the desired QoS policies and attaches to it the specified **PublisherListener** (p. 2006).

[DEPRECATED] When using a listener, prefer the constructor that receives a shared_ptr<**Listener** (p. 1637)> instead of a regular Listener* pointer.

Precondition

The specified QoS policies must be consistent, or the operation will fail

Parameters

the_participant	The DomainParticipant where this **Publisher** (p. 1993) will be created
the_qos	The PublisherQos
the_listener	The **PublisherListener** (p. 2006)
mask	Indicates which status updates the listener will receive

Exceptions

dds::core::Error *(p. 1539)*	If there is any error creating the **Publisher** (p. 1993)

8.267.2.3 Publisher() [3/3]

```
dds::pub::Publisher::Publisher (
            const  ::dds::domain::DomainParticipant & the_participant,
            const  dds::pub::qos::PublisherQos & the_qos,
            std::shared_ptr<  Listener > the_listener,
```

```
            const dds::core::status::StatusMask & mask = dds::core::status::StatusMask::all() )
[inline]
```

Create a **Publisher** (p. 1993) with the desired QoS policies and attaches to it the specified **PublisherListener** (p. 2006).

Precondition

The specified QoS policies must be consistent, or the operation will fail

Parameters

the_participant	The DomainParticipant where this **Publisher** (p. 1993) will be created
the_qos	The PublisherQos
the_listener	A shared_ptr to the listener. See **set_listener()** (p. 1999) for more information.
mask	Indicates which status updates the listener will receive

Exceptions

dds::core::Error *(p. 1539)*	If there is any error creating the **Publisher** (p. 1993)

8.267.3 Member Function Documentation

8.267.3.1 qos() [1/2]

```
const dds::pub::qos::PublisherQos dds::pub::Publisher::qos ( ) const [inline]
```

Gets the PublisherQos.

8.267.3.2 qos() [2/2]

```
void dds::pub::Publisher::qos (
            const dds::pub::qos::PublisherQos & the_qos ) [inline]
```

Set the PublisherQos.

This operation modifies the QoS of the **dds::pub::Publisher** (p. 1993).

The **dds::core::policy::GroupData** (p. 1591), **dds::core::policy::Partition** (p. 1923) and **dds::core::policy::Entity↩ Factory** (p. 1527) can be changed. The other policies are immutable.

Parameters

the_qos	<<***in***>> *(*p. *355)* **dds::pub::qos::PublisherQos** (p. 2007) to be set to. Policies must be consistent. Immutable policies cannot be changed after **dds::pub::Publisher** (p. 1993) is enabled. The special value PUBLISHER_QOS_DEFAULT can be used to indicate that the QoS of the **dds::pub::Publisher** (p. 1993) should be changed to match the current default **dds::pub::qos::PublisherQos** (p. 2007) set in the **dds::domain::DomainParticipant** (p. 1326).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::ImmutablePolicyError** (p. 1609), or **dds::core::InconsistentPolicyError** (p. 1610).

See also

dds::pub::qos::PublisherQos (p. 2007) for rules on consistency among QoS

set_qos (abstract) (p. 1522)

Operations Allowed in Listener Callbacks (p. 1639)

8.267.3.3 operator<<()

```
Publisher & dds::pub::Publisher::operator<< (
            const  dds::pub::qos::PublisherQos & the_qos )  [inline]
```

Sets the PublisherQos.

8.267.3.4 operator>>()

```
Publisher & dds::pub::Publisher::operator>> (
             dds::pub::qos::PublisherQos & the_qos )  [inline]
```

Gets the PublisherQos.

8.267.3.5 default_datawriter_qos() [1/2]

```
Publisher & dds::pub::Publisher::default_datawriter_qos (
            const  dds::pub::qos::DataWriterQos & dwqos )  [inline]
```

Sets the default DataWriterQos.

8.267.3.6 default_datawriter_qos() [2/2]

```
dds::pub::qos::DataWriterQos dds::pub::Publisher::default_datawriter_qos ( ) const  [inline]
```

Gets the default DataWriterQos.

The retrieved DataWriterQos will match the set of values specified on the last call to **default_datawriter_qos(const dds::pub::qos::DataWriterQos&)** (p. 1998). If the application never called that function, the default value comes from the **dds::domain::DomainParticipant** (p. 1326).

8.267.3.7 listener() [1/2]

```
void dds::pub::Publisher::listener (
             Listener * plistener,
            const  dds::core::status::StatusMask & event_mask )  [inline]
```

Sets the **Publisher** (p. 1993) listener.

[DEPRECATED] The use of **set_listener()** (p. 1999) is recommended. Unlike this function, set_listener receives a shared_ptr which simplifies the management of listener's lifecycle.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

set_listener (abstract) (p. 1523)

Parameters

plistener	the topic listener
event_mask	Changes of communication status to be invoked on the listener

8.267.3.8 listener() [2/2]

```
Listener * dds::pub::Publisher::listener ( ) const  [inline]
```

Returns the listener currently associated with this **Publisher** (p. 1993).

[DEPRECATED] Use get_listener instead of this function.

If there's no listener it returns NULL.

8.267.3.9 set_listener() [1/2]

```
void dds::pub::Publisher::set_listener (
            std::shared_ptr<  Listener > the_listener,
            const  dds::core::status::StatusMask & event_mask )  [inline]
```

Sets the listener associated with this publisher.

The **Publisher** (p. 1993) will hold a `shared_ptr` to the listener argument, ensuring that it is not deleted at least until this **Publisher** (p. 1993) is deleted or the listener is reset with `set_listener(nullptr)`.

Warning

It's recommended that the listener implementation doesn't hold a permanent reference to the **Publisher** (p. 1993). If it does, the application needs to manually reset the listener or manually close this **Publisher** (p. 1993) to ensure that there is no cycle that prevents the destruction of these two objects.

Parameters

the_listener	A shared pointer to the listener to receive status updates or `nullptr` to reset the current listener and stop receiving status updates.
event_mask	A mask that indicates which status updates will be notified to the listener

8.267.3.10 set_listener() [2/2]

```
void dds::pub::Publisher::set_listener (
            std::shared_ptr<  Listener > the_listener )  [inline]
```

Sets the listener associated with this publisher.

If `the_listener` is not `nullptr`, this overload is equivalent to:

```
publisher.set_listener(the_listener,
dds::core::status::StatusMask::all());
```

If `the_listener` is `nullptr`, it is equivalent to:

```
publisher.set_listener(nullptr, dds::core::status::StatusMask::none());
```

Parameters

the_listener	A shared pointer to the listener to receive status updates or `nullptr` to reset the current listener and stop receiving status updates.

8.267.3.11 get_listener()

```
std::shared_ptr< Listener > dds::pub::Publisher::get_listener ( ) const [inline]
```

Returns the listener currently associated with this **Publisher** (p. 1993).

Returns

The shared pointer to the current listener or `nullptr` if there is currently no listener associated to this entity.

8.267.3.12 wait_for_acknowledgments()

```
void dds::pub::Publisher::wait_for_acknowledgments (
            const dds::core::Duration & max_wait ) [inline]
```

Blocks the calling thread until all data written by this **Publisher** (p. 1993)'s reliable DataWriters is acknowledged, or until timeout expires.

This operation blocks the calling thread until either all data written by the reliable DataWriters entities is acknowledged by (a) all reliable **dds::sub::DataReader** (p. 976) entities that are matched and alive and (b) by all required subscriptions, or until the duration specified by the `max_wait` parameter elapses, whichever happens first. A successful completion indicates that all the samples written have been acknowledged; a timeout indicates that max_wait elapsed before all the data was acknowledged.

Note that if a thread is blocked in the call to this operation on a **dds::pub::Publisher** (p. 1993) and a different thread writes new samples on any of the reliable DataWriters that belong to this **Publisher** (p. 1993), the new samples must be acknowledged before unblocking the thread that is waiting on this operation.

If none of the **dds::pub::DataWriter** (p. 1138) instances have **dds::core::policy::Reliability** (p. 2149) kind set to RELIABLE, the operation will complete successfully.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::NotEnabledError** (p. 1869), **dds::core::TimeoutError** (p. 2473)

8.267.3.13 participant()

```
const dds::domain::DomainParticipant & dds::pub::Publisher::participant ( ) const [inline]
```

Gets the DomainParticipant to which this **Publisher** (p. 1993) belongs.

8.267.3.14 wait_for_asynchronous_publishing()

```
void wait_for_asynchronous_publishing (
            const  dds::core::Duration & max_wait )
```

<<***extension***>> *(p. 354)* Blocks the calling thread until asynchronous sending is complete.

This operation blocks the calling thread (up to `max_wait`) until all data written by the asynchronous **dds::pub::DataWriter** (p. 1138) entities is sent and acknowledged (if reliable) by all matched **dds::sub::DataReader** (p. 976) entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; if it times out, this indicates that `max_wait` elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort **dds::sub::DataReader** (p. 976) is complete in addition to what **dds::pub::Publisher::wait_for_acknowledgments** (p. 2001) provides.

If none of the **dds::pub::DataWriter** (p. 1138) instances have **rti::core::policy::PublishMode::kind** (p. 2016) set to **rti::core::policy::PublishModeKind_def::ASYNCHRONOUS** (p. 2019), the operation will complete immediately .

Parameters

max_wait	<<***in***>> *(p. 355)* Specifies maximum time to wait for acknowledgements **dds::core::Duration** (p. 1448).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::NotEnabledError** (p. 1869), **dds::core::TimeoutError** (p. 2473)

8.267.3.15 close_contained_entities()

```
void close_contained_entities ( )
```

<<***extension***>> *(p. 354)* Closes all the entities created from this **Publisher** (p. 1993)

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Calling this function explicitly is not necessary to close a **Publisher** (p. 1993).

Deletes all contained **dds::pub::DataWriter** (p. 1138) objects. Once **dds::pub::Publisher::close_contained_entities** (p. 2002) completes successfully, the application may delete the **dds::pub::Publisher** (p. 1993), knowing that it has no contained **dds::pub::DataWriter** (p. 1138) objects.

The operation will fail with **dds::core::PreconditionNotMetError** (p. 1941) if any of the contained entities is in a state where it cannot be deleted.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::PreconditionNotMetError** (p. 1941).

8.267.4 Friends And Related Function Documentation

8.267.4.1 find_publishers() [1/2]

```
template<typename PublisherForwardIterator >
uint32_t find_publishers (
            const  dds::domain::DomainParticipant participant,
            PublisherForwardIterator begin,
            uint32_t max_size )  [related]
```

<<***extension***>> *(p. 354)* Retrieve all of the **dds::pub::Publisher** (p. 1993) created from this **dds::domain::Domain↩ Participant** (p. 1326)

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Template Parameters

PublisherForwardIterator	A forward iterator whose value type is **dds::pub::Publisher** (p. 1993)

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Publishers belong to
begin	A forward iterator to the position in the destination container to insert the found Publishers into
max_size	The maximum number of Publishers to add

Returns

The number of found Publishers

See also

Looking up DataWriters (p. 314)

8.267.4.2 find_publishers() [2/2]

```
template<typename PublisherBackInsertIterator >
uint32_t find_publishers (
             const  dds::domain::DomainParticipant participant,
             PublisherBackInsertIterator begin )  [related]
```

Retrieve all the **dds::pub::Publisher** (p. 1993) created from this **dds::domain::DomainParticipant** (p. 1326).

Template Parameters

PublisherBackInsertIterator	A back-inserting iterator whose value type is **dds::pub::Publisher** (p. 1993)

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Publishers belong to
begin	A back-inserting iterator to the position in the destination container to insert the found Publishers in

Returns

The number of found Publishers

See also

Looking up DataWriters (p. 314)

8.267.4.3 find_publisher()

```
dds::pub::Publisher find_publisher (
             const  dds::domain::DomainParticipant participant,
             const std::string & publisher_name )  [related]
```

<<***extension***>> *(p. 354)* Looks up a **dds::pub::Publisher** (p. 1993) by its entity name within the **dds::domain::DomainParticipant** (p. 1326).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

Every **dds::pub::Publisher** (p. 1993) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 532).

This operation retrieves a **dds::pub::Publisher** (p. 1993) within the **dds::domain::DomainParticipant** (p. 1326) given the entity's name. If there are several **dds::pub::Publisher** (p. 1993) with the same name within the **dds::domain::DomainParticipant** (p. 1326), this function returns the first matching occurrence.

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) to look for the **dds::pub::Publisher** (p. 1993) in.
publisher_name	Entity name of the **Publisher** (p. 1993)

Returns

The first **Publisher** (p. 1993) found with the specified name or **dds::core::null** (p. 438) if it is not found.

See also

Looking up DataWriters (p. 314)

8.267.4.4 implicit_publisher()

```
dds::pub::Publisher implicit_publisher (
            const  dds::domain::DomainParticipant & dp )  [related]
```

<<***extension***>> *(p. 354)* Get the implicit **dds::pub::Publisher** (p. 1993) for a given **dds::domain::Domain↩ Participant** (p. 1326).

Note

This is a standalone function in the namespace **rti::pub** (p. 736)

If an implicit **Publisher** (p. 1993) does not already exist, this creates one.

The implicit **Publisher** (p. 1993) is created with default **dds::pub::qos::PublisherQos** (p. 2007) and no listener. When a DomainParticipant is deleted, if there are no attached **dds::pub::DataWriter** (p. 1138) that belong to the implicit **Publisher** (p. 1993), the implicit **Publisher** (p. 1993) will be implicitly deleted.

MT Safety:

UNSAFE. It is not safe to create an implicit **Publisher** (p. 1993) while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_publisher_qos** (p. 1342).

Parameters

dp	The DomainParticipant that the implicit publisher belongs to

Returns

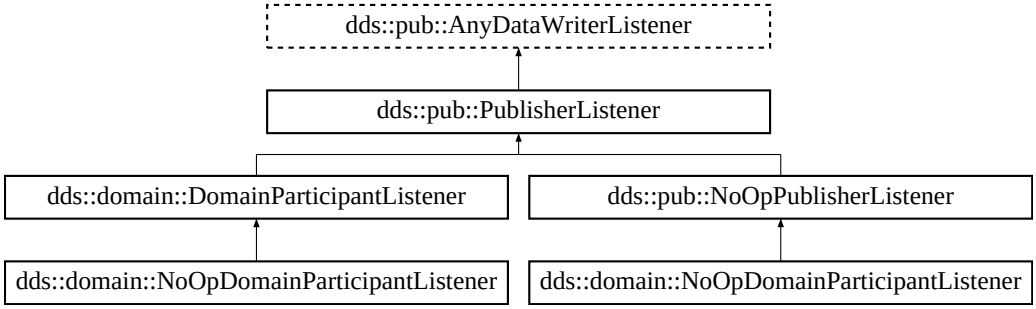
The implicit publisher

8.268 dds::pub::PublisherListener Class Reference

The listener to notify status changes for a **dds::pub::Publisher** (p. 1993).

`#include <dds/pub/PublisherListener.hpp>`

Inheritance diagram for dds::pub::PublisherListener:



Additional Inherited Members

8.268.1 Detailed Description

The listener to notify status changes for a **dds::pub::Publisher** (p. 1993).

A **PublisherListener** (p. 2006) contains exactly the same methods as a **AnyDataWriterListener** (p. 821), since it can receive status changes from any of its contained DataWriters.

Entity:

dds::pub::Publisher (p. 1993)

Status:

dds::core::status::StatusMask::liveliness_lost() (p. 2383), **dds::core::status::LivelinessLostStatus** (p. 1655);
dds::core::status::StatusMask::offered_deadline_missed() (p. 2379), **dds::core::status::OfferedDeadlineMissedStatus** (p. 1871);
dds::core::status::StatusMask::offered_incompatible_qos() (p. 2380), **dds::core::status::OfferedIncompatibleQosStatus** (p. 1873);
dds::core::status::StatusMask::publication_matched() (p. 2384), **dds::core::status::PublicationMatchedStatus** (p. 1990);
dds::core::status::StatusMask::reliable_reader_activity_changed() (p. 2385), **rti::core::status::ReliableReaderActivityChangedStatus** (p. 2157);
dds::core::status::StatusMask::reliable_writer_cache_changed() (p. 2385), **rti::core::status::ReliableWriterCacheChangedStatus** (p. 2159)

See also

Listener (p. 1637)

Status Kinds (p. 429)

Operations Allowed in Listener Callbacks (p. 1639)

NoOpPublisherListener (p. 1852)

8.269 dds::pub::qos::PublisherQos Class Reference

<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::pub::Publisher** (p. 1993) supports

`#include <dds/pub/qos/PublisherQos.hpp>`

Public Member Functions

- template<typename POLICY >
  const POLICY & **policy** () const

  Gets a QoS policy by const reference.
- template<typename POLICY >
  POLICY & **policy** ()

  Gets a QoS policy by reference.
- template<typename Policy >
  PublisherQos & **policy** (const Policy &p)

  Sets a policy.
- template<typename Policy >
  PublisherQos & **operator**<< (const Policy &p)

  Sets a policy.
- template<typename Policy >
  const **PublisherQos** & **operator**>> (Policy &p) const

  Copies the values of a policy.

Related Functions

(Note that these are not member functions.)

- void **swap** (PublisherQosImpl &left, PublisherQosImpl &right) OMG_NOEXCEPT

  *Swap the contents of two **PublisherQos** (p. 2007) objects.*
- std::string **to_string** (const **PublisherQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::Qos**↩**PrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the **dds::pub::qos::PublisherQos** (p. 2007)*
- std::string **to_string** (const **PublisherQos** &qos, const **PublisherQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the **dds::pub::qos::PublisherQos** (p. 2007)*
- std::string **to_string** (const **PublisherQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti**↩**::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the **dds::pub::qos::PublisherQos** (p. 2007)*
- std::ostream & **operator**<< (std::ostream &out, const rti::pub::qos::PublisherQos &qos)

  <<***extension***>> *(p. 354) Prints a **dds::pub::qos::PublisherQos** (p. 2007) to an output stream.*

8.269.1 Detailed Description

<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::pub::Publisher** (p. 1993) supports

8.269.2 PublisherQos Policies

A **PublisherQos** (p. 2007) contains the following policies:

- **dds::core::policy::Presentation** (p. 1942),
- **dds::core::policy::Partition** (p. 1923),
- **dds::core::policy::GroupData** (p. 1591),
- **dds::core::policy::EntityFactory** (p. 1527),
- **rti::core::policy::AsynchronousPublisher** (p. 832),
- **rti::core::policy::EntityName** (p. 1530)

To get or set policies use the **policy()** (p. 2009) getters and setters or operator << (see **examples** (p. 603)).

You must set certain members in a consistent manner:

length of **dds::core::policy::GroupData::value** (p. 1593) <= **rti::core::policy::DomainParticipantResourceLimits**↩ **::publisher_group_data_max_length** (p. 1415)

length of **dds::core::policy::Partition::name** (p. 1926) <= **rti::core::policy::DomainParticipantResourceLimits**↩ **::max_partitions** (p. 1417)

combined number of characters (including terminating 0) in **dds::core::policy::Partition::name** (p. 1926) <= **rti**↩ **::core::policy::DomainParticipantResourceLimits::max_partition_cumulative_characters** (p. 1417)

If any of the above are not true, **dds::pub::Publisher::qos(const dds::pub::qos::PublisherQos&)** (p. 1997) and dds↩ ::pub::Publisher::set_qos_with_profile will fail with **dds::core::InconsistentPolicyError** (p. 1610) and the **dds::pub::**↩ **Publisher** (p. 1993) constructors will fail with **dds::core::Error** (p. 1539)

See also

Qos Use Cases (p. 602)

8.269.3 Member Function Documentation

8.269.3.1 policy() [1/3]

```
template<typename POLICY >
const POLICY & dds::pub::qos::PublisherQos::policy ( ) const
```

Gets a QoS policy by const reference.

Template Parameters

Policy	One of the **PublisherQos Policies** (p. 2008)

See also

Setting Qos Values (p. 603)

8.269.3.2 policy() [2/3]

```
template<typename POLICY >
POLICY & dds::pub::qos::PublisherQos::policy ( )
```

Gets a QoS policy by reference.

Template Parameters

Policy	One of the **PublisherQos Policies** (p. 2008)

See also

Setting Qos Values (p. 603)

8.269.3.3 policy() [3/3]

```
template<typename Policy >
PublisherQos & dds::pub::qos::PublisherQos::policy (
            const Policy & p )  [inline]
```

Sets a policy.

See also

policy() (p. 2009)

Setting Qos Values (p. 603)

8.269.3.4 operator<<()

```
template<typename Policy >
PublisherQos & dds::pub::qos::PublisherQos::operator<< (
            const Policy & p )  [inline]
```

Sets a policy.

See also

policy() (p. 2009)

Setting Qos Values (p. 603)

8.269.3.5 operator>>()

```
template<typename Policy >
const  PublisherQos & dds::pub::qos::PublisherQos::operator>> (
            Policy & p ) const  [inline]
```

Copies the values of a policy.

Parameters

p	The destination where to copy the current value of the Policy

See also

policy() (p. 2009)

Setting Qos Values (p. 603)

8.269.4 Friends And Related Function Documentation

8.269.4.1 swap()

```
void swap (
            PublisherQosImpl & left,
            PublisherQosImpl & right )  [related]
```

Swap the contents of two **PublisherQos** (p. 2007) objects.

Parameters

left	A **PublisherQos** (p. 2007)
right	The other **PublisherQos** (p. 2007)

8.269.4.2 to_string() [1/3]

```
std::string to_string (
            const  PublisherQos & qos,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::pub::qos::PublisherQos** (p. 2007)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
PublisherQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for PublisherQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
PublisherQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

8.269.4.3 to_string() [2/3]

```
std::string to_string (
            const  PublisherQos & qos,
            const  PublisherQos & base,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(*p. *354)* Obtains a string representation of the **dds::pub::qos::PublisherQos** (p. 2007)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

8.269.4.4 to_string() [3/3]

```
std::string to_string (
            const  PublisherQos & qos,
            const  rti::core::qos_print_all_t & qos_print_all,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(*p. *354)* Obtains a string representation of the **dds::pub::qos::PublisherQos** (p. 2007)

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
`to_string(qos, rti::core::qos_print_all);`

Returns

The string representation of the qos

8.269.4.5 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const rti::pub::qos::PublisherQos & qos )  [related]
```

<<***extension***>> *(*p. *354)* Prints a **dds::pub::qos::PublisherQos** (p. 2007) to an output stream.

8.270 rti::core::policy::PublishMode Class Reference

<<***extension***>> *(*p. *116)* Specifies whether a **dds::pub::DataWriter** (p. 1138) sends data synchronously or asynchronously.

`#include <rti/core/policy/CorePolicy.hpp>`

Public Member Functions

- **PublishMode** ()

  *Creates a **PublishMode** (*p. *2013) qos policy of synchronous kind.*
- **PublishMode** & **kind** (const **rti::core::policy::PublishModeKind** the_kind)

  Sets the publish mode (synchronous or asynchronous) for a DataWriter.
- **rti::core::policy::PublishModeKind kind** () const

  Gets the publish mode.
- std::string **flow_controller_name** () const

  Gets the flow controller name.
- **PublishMode** & **flow_controller_name** (const std::string &name)

  Sets the flow controller name associated with a DataWriter.
- int32_t **priority** () const

  Sets the priority of all samples written by a DataWriter.
- **PublishMode** & **priority** (int32_t value)

  Gets the priority.

Static Public Member Functions

- static **PublishMode Synchronous** ()

  *Creates a **PublishMode** (*p. *2013) qos policy of synchronous kind.*
- static **PublishMode Asynchronous** ()

  *Creates a **PublishMode** (*p. *2013) qos policy of asynchronous kind with default flow controller and undefined priority.*
- static **PublishMode Asynchronous** (const std::string & **flow_controller_name**)

  *Creates a **PublishMode** (*p. *2013) qos policy of asynchronous kind with a specific flow controller and undefined priority.*
- static **PublishMode Asynchronous** (const std::string & **flow_controller_name**, int32_t **priority**)

  *Creates a **PublishMode** (*p. *2013) qos policy of asynchronous kind with specific flow controller and priority.*

8.270.1 Detailed Description

<<***extension***>> *(*p. *116)* Specifies whether a **dds::pub::DataWriter** (p. 1138) sends data synchronously or asynchronously.

The publishing mode of a **dds::pub::DataWriter** (p. 1138) determines whether data is written synchronously in the context of the user thread when calling **dds::pub::DataWriter::write()** (p. 1148) or asynchronously in the context of a separate thread internal to the middleware.

Each **dds::pub::Publisher** (p. 1993) spawns a single asynchronous publishing thread (**rti::core::policy::AsynchronousPublisher::thread** (p. 835)) to serve all its asynchronous **dds::pub::DataWriter** (p. 1138) instances.

See also

rti::core::policy::AsynchronousPublisher (p. 832)

dds::core::policy::History (p. 1602)

rti::pub::FlowController (p. 1572)

Entity:

dds::pub::DataWriter (p. 1138)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.270.2 Usage

The fastest way for RTI Connext to send data is for the user thread to execute the middleware code that actually sends the data itself. However, there are times when user applications may need or want an internal middleware thread to send the data instead.

When data is written asynchronously, a **rti::pub::FlowController** (p. 1572), identified by `flow_controller_name`, can be used to shape the network traffic. Shaping a data flow usually means limiting the maximum data rates at which the middleware will send data for a **dds::pub::DataWriter** (p. 1138). The flow controller will buffer any excess data and only send it when the send rate drops below the maximum rate. The flow controller's properties determine when the asynchronous publishing thread is allowed to send data and how much.

Asynchronous publishing may increase latency, but offers the following advantages:

- The **dds::pub::DataWriter::write()** (p. 1148) call does not make any network calls and is therefore faster and more deterministic. This becomes important when the user thread is executing time-critical code.

- When data is written in bursts or when sending large data types as multiple fragments, a flow controller can throttle the send rate of the asynchronous publishing thread to avoid flooding the network.

- Asynchronously written samples for the same destination will be coalesced into a single network packet which reduces bandwidth consumption.

The maximum number of samples that will be coalesced depends on **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789) (each sample requires at least 2-4 gather-send buffers). Performance can be improved by increasing **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789). Note that the maximum value is operating system dependent.

The middleware must queue samples until they can be sent by the asynchronous publishing thread (as determined by the corresponding **rti::pub::FlowController** (p. 1572)). The number of samples that will be queued is determined by the **dds::core::policy::History** (p. 1602). When using **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607), only the most recent **dds::core::policy::History::depth** (p. 1605) samples are kept in the queue. Once unsent samples are removed from the queue, they are no longer available to the asynchronous publishing thread and will therefore never be sent.

8.270.3 Constructor & Destructor Documentation

8.270.3.1 PublishMode()

```
rti::core::policy::PublishMode::PublishMode ( )
```

Creates a **PublishMode** (p. 2013) qos policy of synchronous kind.

8.270.4 Member Function Documentation

8.270.4.1 Synchronous()

```
static PublishMode rti::core::policy::PublishMode::Synchronous ( ) [static]
```

Creates a **PublishMode** (p. 2013) qos policy of synchronous kind.

8.270.4.2 Asynchronous() [1/3]

```
static PublishMode rti::core::policy::PublishMode::Asynchronous ( ) [static]
```

Creates a **PublishMode** (p. 2013) qos policy of asynchronous kind with default flow controller and undefined priority.

8.270.4.3 Asynchronous() [2/3]

```
static PublishMode rti::core::policy::PublishMode::Asynchronous (
            const std::string & flow_controller_name ) [static]
```

Creates a **PublishMode** (p. 2013) qos policy of asynchronous kind with a specific flow controller and undefined priority.

8.270.4.4 Asynchronous() [3/3]

```
static PublishMode rti::core::policy::PublishMode::Asynchronous (
            const std::string & flow_controller_name,
            int32_t priority ) [static]
```

Creates a **PublishMode** (p. 2013) qos policy of asynchronous kind with specific flow controller and priority.

8.270.4.5 kind() [1/2]

```
PublishMode & rti::core::policy::PublishMode::kind (
            const rti::core::policy::PublishModeKind the_kind )
```

Sets the publish mode (synchronous or asynchronous) for a DataWriter.

[default] rti::core::policy::PublishModeKind_def::SYNCHRONOUS (p. 2019)

Categories

Immutable (p. 243), **Flow Control** (p. 188)

8.270.4.6 kind() [2/2]

```
rti::core::policy::PublishModeKind rti::core::policy::PublishMode::kind ( ) const
```

Gets the publish mode.

8.270.4.7 flow_controller_name() [1/2]

```
std::string rti::core::policy::PublishMode::flow_controller_name ( ) const
```

Gets the flow controller name.

8.270.4.8 flow_controller_name() [2/2]

```
PublishMode & rti::core::policy::PublishMode::flow_controller_name (
            const std::string & name )
```

Sets the flow controller name associated with a DataWriter.

The following builitin values are supported:

- **rti::pub::FlowController::DEFAULT_NAME** (p. 58)
- **rti::pub::FlowController::FIXED_RATE_NAME** (p. 59)
- **rti::pub::FlowController::ON_DEMAND_NAME** (p. 60)

See also

rti::pub::FlowController::FlowController(dds::domain::DomainParticipant,const std::string&,const FlowControllerProperty&) (p. 1573)

[default] rti::pub::FlowController::DEFAULT_NAME (p. 58)

Categories

Immutable (p. 243), **Flow Control** (p. 188)

8.270.4.9 priority() [1/2]

```
int32_t rti::core::policy::PublishMode::priority ( ) const
```

Sets the priority of all samples written by a DataWriter.

A positive integer value designating the relative priority of the **dds::pub::DataWriter** (p. 1138), used to determine the transmission order of pending writes.

Use of publication priorities requires the asynchronous publisher (**rti::core::policy::PublishModeKind_def::ASYNCHRONOUS** (p. 2019)) with **rti::pub::FlowControllerProperty::scheduling_policy** (p. 1579) set to FlowControllerSchedulingPolicy_def::HIGHEST_PRIORITY_FIRST.

Larger numbers have higher priority.

For multi-channel DataWriters, if the publication priority of any channel is set to any value other than PUBLICATION_PRIORITY_UNDEFINED, then the channel's priority will take precedence over that of the DataWriter.

For multi-channel DataWriters, if the publication priority of any channel is PUBLICATION_PRIORITY_UNDEFINED, then the channel will inherit the publication priority of the DataWriter.

If the publicaton priority of the DataWriter, and of any channel of a multi-channel DataWriter, are PUBLICATION_↩ PRIORITY_UNDEFINED, then the priority of the DataWriter or DataWriter channel will be assigned the lowest priority value.

If the publication priority of the DataWriter is PUBLICATION_PRIORITY_AUTOMATIC, then the DataWriter will be assigned the priority of the largest publication priority of all samples in the DataWriter.

The publication priority of each sample can be set in the **rti::pub::WriteParams** (p. 2642) of the **dds::pub::Data↩ Writer::write(const T&,rti::pub::WriteParams&)** (p. 1179) function.

For dispose and unregister samples, use the **rti::pub::WriteParams** (p. 2642) of **dds::pub::DataWriter::dispose↩ _instance(rti::pub::WriteParams&)** (p. 1173) and **dds::pub::DataWriter::unregister_instance(rti::pub::Write↩ Params&)** (p. 1173).

[default] PUBLICATION_PRIORITY_UNDEFINED

[range] [-1, MAX_INT]

Categories

Immutable (p. 243), **Flow Control** (p. 188)

8.270.4.10 priority() [2/2]

```
PublishMode & rti::core::policy::PublishMode::priority (
            int32_t value )
```

Gets the priority.

8.271 rti::core::policy::PublishModeKind_def Struct Reference

<<***extension***>> *(p. 354)* The enumeration for **PublishMode** (p. 2013) kinds

`#include <PolicyKind.hpp>`

Public Types

- enum **type** {
  SYNCHRONOUS ,
  ASYNCHRONOUS }

  The underlying `enum` type.

8.271.1 Detailed Description

<<***extension***>> *(p. 354)* The enumeration for **PublishMode** (p. 2013) kinds

8.271.2 Member Enumeration Documentation

8.271.2.1 type

enum **rti::core::policy::PublishModeKind_def::type**

The underlying enum type.

Enumerator

SYNCHRONOUS	Indicates to send data synchronously. If **rti::core::policy::DataWriterProtocol::push_on_write** (p. 1213) is true, data is sent immediately in the context of **dds::pub::DataWriter::write()** (p. 1148). As data is sent immediately in the context of the user thread, no flow control is applied. **See also** **rti::core::policy::DataWriterProtocol::push_on_write** (p. 1213) **[default]** for **dds::pub::DataWriter** (p. 1138)
ASYNCHRONOUS	Indicates to send data asynchronously. Configures the **dds::pub::DataWriter** (p. 1138) to delegate the task of data transmission to a separate publishing thread. The **dds::pub::DataWriter::write()** (p. 1148) call does not send the data, but instead schedules the data to be sent later by its associated **dds::pub::Publisher** (p. 1993). Each **dds::pub::Publisher** (p. 1993) uses its dedicated publishing thread (**rti::core::policy::AsynchronousPublisher** (p. 832)) to send data for all its asynchronous DataWriters. For each asynchronous DataWriter, the associated **rti::pub::FlowController** (p. 1572) determines when the publishing thread is allowed to send the data. **dds::pub::DataWriter::wait_for_asynchronous_publishing** (p. 1174) and **dds::pub::Publisher::wait_for_asynchronous_publishing** (p. 2001) enable you to determine when the data has actually been sent. **See also** **rti::pub::FlowController** (p. 1572) **dds::core::policy::History** (p. 1602) **dds::pub::DataWriter::wait_for_asynchronous_publishing** (p. 1174) **dds::pub::Publisher::wait_for_asynchronous_publishing** (p. 2001) **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 1789)

8.272 rti::core::qos_print_all_t Struct Reference

A tag type that selects the to_string overload that prints all the values of a Qos object.

#include <types.hpp>

8.272.1 Detailed Description

A tag type that selects the to_string overload that prints all the values of a Qos object.

The only value of this type is **rti::core::qos_print_all** (p. 439), and is used only as a sentinel to select one of the to_string overloads.

For example:

auto s = to_string(my_reader_qos, ..., rti::core::qos_print_all);

See rti::sub::qos::to_string(...).

See also

rti::sub::qos::to_string

8.273 dds::core::policy::QosPolicyCount Class Reference

<<***value-type***>> *(*p. *350)* Holds a counter for a QosPolicyId

`#include <dds/core/policy/QosPolicyCount.hpp>`

Public Member Functions

- **QosPolicyCount** (**QosPolicyId** the_policy_id, int32_t the_count)
  Creates an instance with the policy ID and its counter.
- **QosPolicyId policy_id** () const
  Gets the policy ID.
- int32_t **count** () const
  Gets the counter.

8.273.1 Detailed Description

<<***value-type***>> *(*p. *350)* Holds a counter for a QosPolicyId

8.273.2 Constructor & Destructor Documentation

8.273.2.1 QosPolicyCount()

```
dds::core::policy::QosPolicyCount::QosPolicyCount (
              QosPolicyId the_policy_id,
             int32_t the_count )  [inline]
```

Creates an instance with the policy ID and its counter.

8.273.3 Member Function Documentation

8.273.3.1 policy_id()

```
QosPolicyId dds::core::policy::QosPolicyCount::policy_id ( ) const  [inline]
```

Gets the policy ID.

8.273.3.2 count()

```
int32_t dds::core::policy::QosPolicyCount::count ( ) const  [inline]
```

Gets the counter.

8.274 rti::core::QosPrintFormat Class Reference

<<***extension***>> (p. 354) <<***value-type***>> (p. 350) A collection of attributes used to configure how QoS will be formatted when converted to strings.

```
#include <rti/core/QosPrintFormat.hpp>
```

Public Member Functions

- **QosPrintFormat** ()
  Initializes the properties with the default values.
- **QosPrintFormat** (unsigned int indent_in, bool print_private_in, bool is_standalone_in)
  Initializes the properties.
- unsigned int **indent** () const
  Get the value of indent.
- **QosPrintFormat** & **indent** (unsigned int value)
  Set the amount of additional indent to be included when converting a QoS to a string.
- bool **print_private** () const
  Get the value of print_private.
- **QosPrintFormat** & **print_private** (bool value)
  Set whether or not to print private Qos fields.
- bool **is_standalone** () const
  Get the value of is_standalone.
- **QosPrintFormat** & **is_standalone** (bool value)
  Set whether or not to print the XML preamble.

8.274.1 Detailed Description

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* A collection of attributes used to configure how QoS will be formatted when converted to strings.

8.274.2 Constructor & Destructor Documentation

8.274.2.1 QosPrintFormat() [1/2]

```
rti::core::QosPrintFormat::QosPrintFormat ( )  [inline]
```

Initializes the properties with the default values.

8.274.2.2 QosPrintFormat() [2/2]

```
rti::core::QosPrintFormat::QosPrintFormat (
             unsigned int indent_in,
             bool print_private_in,
             bool is_standalone_in )  [inline]
```

Initializes the properties.

8.274.3 Member Function Documentation

8.274.3.1 indent() [1/2]

```
unsigned int rti::core::QosPrintFormat::indent ( ) const  [inline]
```

Get the value of indent.

See also

indent(unsigned int value) (p. 2022)

8.274.3.2 indent() [2/2]

```
QosPrintFormat & rti::core::QosPrintFormat::indent (
            unsigned int value )  [inline]
```

Set the amount of additional indent to be included when converting a QoS to a string.

Configures how much additional indent is applied when converting a QoS to a string. This value acts as a total offset on the string, increasing the indent which is applied to all elements by the same amount. With indent set to 0, a string representation of a QoS may appear as:

```
<datareader_qos>
    <durability>
        <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>
    </durability>
</datareader_qos>
```

Setting the indent property to 1, the same QoS would be printed as:

```
    <datareader_qos>
        <durability>
            <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>
        </durability>
    </datareader_qos>
```

I.e., the entire structure is indented.

[default] 0

Parameters

value	The value to set for indent

8.274.3.3 print_private() [1/2]

```
bool rti::core::QosPrintFormat::print_private ( ) const  [inline]
```

Get the value of print_private.

See also

print_private(bool value) (p. 2023)

8.274.3.4 print_private() [2/2]

```
QosPrintFormat & rti::core::QosPrintFormat::print_private (
            bool value )  [inline]
```

Set whether or not to print private Qos fields.

There are some QoS policies which are not intended for external use. By default, these QoS policies are only printed if they are different to the default value for that policy. When true, private policies are treated like any other policy, and printed if they are different to the supplied base QoS. These private policies cannot be parsed from XML, and are always printed as XML comments.

[default] false

Parameters

value	The value to set for print_private.

8.274.3.5 is_standalone() [1/2]

```
bool rti::core::QosPrintFormat::is_standalone ( ) const  [inline]
```

Get the value of is_standalone.

See also

print_private(bool value) (p. 2023)

8.274.3.6 is_standalone() [2/2]

```
QosPrintFormat & rti::core::QosPrintFormat::is_standalone (
            bool value )  [inline]
```

Set whether or not to print the XML preamble.

The default value for this property (false) results in QoS being printed starting from the <ENTITY_qos> tag (where the <ENTITY_qos> tag is, e.g., <domain_participant_qos>, or, <datareader_qos>. This result cannot be directly parsed as valid XML (as it lacks <dds>, <qos_library>, and <qos_profile> tags). If is_standalone is set to true, these additional tags are printed, resulting in valid, parseable XML.

For example, with is_standalone set to false, a Qos may appear as:

```
<datareader_qos>
    <durability>
        <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>
    </durability>
</datareader_qos>
```

Setting is_standalone to true would result in the same Qos being printed as:

```
<?xml version="1.0"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <qos_library name="QosLibrary">
        <qos_profile name="QosProfile">
            <datareader_qos>
                <durability>
                    <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>
                </durability>
            </datareader_qos>
        </qos_profile>
    </qos_library>
</dds>
```

[default] false

Parameters

value	The value to set for is_standalone.

8.275 dds::core::QosProvider Class Reference

*<<**reference-type**>> (p. 351)* The **QosProvider** (p. 2025) class provides a way for a user to control and access the XML QoS profiles that are loaded by RTI Connext

```
#include <dds/core/QosProvider.hpp>
```

Public Member Functions

- **QosProvider** (const std::string &uri, const std::string &profile)

  *Create a **QosProvider** (p. 2025) fetching QoS configuration from the specified URI.*

- **QosProvider** (const std::string &uri)

  *Create a **QosProvider** (p. 2025) fetching QoS configuration from the specified URI.*
- const **dds::domain::qos::DomainParticipantQos participant_qos** ()

  *Get the **dds::domain::qos::DomainParticipantQos** (p. 1384) in the default profile.*
- const **dds::domain::qos::DomainParticipantQos participant_qos** (const std::string &profile)

  Get the DomainParticipantQos from a QoS profile.
- const **dds::topic::qos::TopicQos topic_qos** ()

  *Get the **dds::topic::qos::TopicQos** (p. 2509) in the default profile.*
- const **dds::topic::qos::TopicQos topic_qos** (const std::string &profile)

  *Get the **dds::topic::qos::TopicQos** (p. 2509) from a QoS profile.*
- const **dds::sub::qos::SubscriberQos subscriber_qos** ()

  *Get the **dds::sub::qos::SubscriberQos** (p. 2423) in the default profile.*
- const **dds::sub::qos::SubscriberQos subscriber_qos** (const std::string &profile)

  *Get the **dds::sub::qos::SubscriberQos** (p. 2423) from a QoS profile.*
- const **dds::sub::qos::DataReaderQos datareader_qos** ()

  *Get the **dds::sub::qos::DataReaderQos** (p. 1072) in the default profile.*
- const **dds::sub::qos::DataReaderQos datareader_qos** (const std::string &profile)

  *Get the **dds::sub::qos::DataReaderQos** (p. 1072) from a QoS profile.*
- const **dds::pub::qos::PublisherQos publisher_qos** ()

  *Get the **dds::pub::qos::PublisherQos** (p. 2007) in the default profile.*
- **dds::pub::qos::PublisherQos publisher_qos** (const std::string &profile)

  *Get the **dds::pub::qos::PublisherQos** (p. 2007) from a QoS profile.*
- const **dds::pub::qos::DataWriterQos datawriter_qos** ()

  *Get the **dds::pub::qos::DataWriterQos** (p. 1226) from the default profile.*
- const **dds::pub::qos::DataWriterQos datawriter_qos** (const std::string &profile)

  *Get the **dds::pub::qos::DataWriterQos** (p. 1226) from a QoS profile.*
- **dds::domain::qos::DomainParticipantFactoryQos participant_factory_qos** () const

  *<<**extension**>> (p. 354) Get the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378) in the default profile.*
- **dds::domain::qos::DomainParticipantFactoryQos participant_factory_qos** (const std::string &profile) const

  *<<**extension**>> (p. 354) Get the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378) from a QoS profile*
- const **dds::topic::qos::TopicQos topic_qos_w_topic_name** (const std::string &profile, const std::string &topic_name) const

  *<<**extension**>> (p. 354) Get the **dds::topic::qos::TopicQos** (p. 2509) for a topic name in a given profile.*
- const **dds::topic::qos::TopicQos topic_qos_w_topic_name** (const std::string &topic_name) const

  *<<**extension**>> (p. 354) Get the **dds::topic::qos::TopicQos** (p. 2509) for a topic name in the default profile.*
- const **dds::sub::qos::DataReaderQos datareader_qos_w_topic_name** (const std::string &profile, const std::string &topic_name) const

  *<<**extension**>> (p. 354) Get the **dds::sub::qos::DataReaderQos** (p. 1072) for a topic name in a given profile.*
- const **dds::sub::qos::DataReaderQos datareader_qos_w_topic_name** (const std::string &topic_name) const

  *<<**extension**>> (p. 354) Get the **dds::sub::qos::DataReaderQos** (p. 1072) for a topic name in the default profile.*
- const **dds::pub::qos::DataWriterQos datawriter_qos_w_topic_name** (const std::string &profile, const std::string &topic_name) const

  *<<**extension**>> (p. 354) Get the **dds::pub::qos::DataWriterQos** (p. 1226) for a topic name.*
- const **dds::pub::qos::DataWriterQos datawriter_qos_w_topic_name** (const std::string &topic_name) const

  *<<**extension**>> (p. 354) Get the **dds::pub::qos::DataWriterQos** (p. 1226) for a topic name.*
- void **default_library** (const std::string &library_name)

  *<<**extension**>> (p. 354) Set the default library for this **QosProvider** (p. 2025).*

- void **default_profile** (const std::string &profile_name)

  <<***extension***>> *(p. 354) Set the default profile for this **QosProvider** (p. 2025).*
- **rti::core::optional_value**< std::string > **default_library** () const

  <<***extension***>> *(p. 354) Get the default library associated with this **QosProvider** (p. 2025).*
- **rti::core::optional_value**< std::string > **default_profile** () const

  <<***extension***>> *(p. 354) Get the default profile associated with this **QosProvider** (p. 2025).*
- **rti::core::optional_value**< std::string > **default_profile_library** () const

  <<***extension***>> *(p. 354) Get the default profile library associated with this **QosProvider** (p. 2025).*
- **dds::core::StringSeq qos_profile_libraries** () const

  <<***extension***>> *(p. 354) Get a list of the QoS profile libraries loaded by this **QosProvider** (p. 2025).*
- **dds::core::StringSeq qos_profiles** (const std::string &library_name) const

  <<***extension***>> *(p. 354) Get a list of the QoS profiles located in the default library of this **QosProvider** (p. 2025).*
- bool **profiles_loaded** () const

  <<***extension***>> *(p. 354) Check if the profiles are loaded by this **QosProvider** (p. 2025)*
- const **dds::core::xtypes::DynamicType** & **type** (const std::string &type_name) const

  <<***extension***>> *(p. 354) Load a type*
- **rti::core::QosProviderParams provider_params** () const

  <<***extension***>> *(p. 354) Get the **QosProvider** (p. 2025) params for this **QosProvider** (p. 2025).*
- void **provider_params** (const **rti::core::QosProviderParams** &provider_params)

  <<***extension***>> *(p. 354) Set the **rti::core::QosProviderParams** (p. 2049) for this **QosProvider** (p. 2025).*
- void **load_profiles** ()

  <<***extension***>> *(p. 354) Load the XML QoS profiles.*
- void **reload_profiles** ()

  <<***extension***>> *(p. 354) Reload the XML QoS profiles.*
- void **unload_profiles** ()

  <<***extension***>> *(p. 354) Unload the XML QoS profiles.*
- **dds::domain::DomainParticipant create_participant_from_config** (const std::string &config_name, const **rti::domain::DomainParticipantConfigParams** &params= **rti::domain::DomainParticipantConfigParams**())

  <<***extension***>> *(p. 354) Creates a **dds::domain::DomainParticipant** (p. 1326) given its configuration name from a description provided in an XML configuration file that has been loaded by this **QosProvider** (p. 2025).*

Static Public Member Functions

- static **QosProvider Default** ()

  *Get the default **QosProvider** (p. 2025).*
- static void **reset_default** ()

  *Reset the default settings of the default **QosProvider** (p. 2025).*
- static **rti::core::QosProviderParams default_provider_params** ()

  <<***extension***>> *(p. 354) (deprecated) Get the **rti::core::QosProviderParams** (p. 2049) for the default **QosProvider** (p. 2025)*
- static void **default_provider_params** (const **rti::core::QosProviderParams** &params)

  <<***extension***>> *(p. 354) (deprecated) Set the **rti::core::QosProviderParams** (p. 2049) for the default **QosProvider** (p. 2025)*

Related Functions

(Note that these are not member functions.)

- const char * **USE_DDS_DEFAULT_QOS_PROFILE**

  Special value to select the default QoS profile.
- dds::core::detail::QosProvider **create_qos_provider_ex** (const **rti::core::QosProviderParams** &params)

  <<***extension***>> *(p. 354) Creates a* ***QosProvider*** *(p. 2025) with parameters*
- QosProviderParams **default_qos_provider_params** ()

  <<***extension***>> *(p. 354) Get the* ***rti::core::QosProviderParams*** *(p. 2049) for the default* ***QosProvider*** *(p. 2025)*
- void **default_qos_provider_params** (const **rti::core::QosProviderParams** &params)

  <<***extension***>> *(p. 354) Set the* ***rti::core::QosProviderParams*** *(p. 2049) for the default* ***QosProvider*** *(p. 2025)*

8.275.1 Detailed Description

<<***reference-type***>> *(p. 351)* The **QosProvider** (p. 2025) class provides a way for a user to control and access the XML QoS profiles that are loaded by RTI Connext

A **QosProvider** (p. 2025) is created with a URI that identifies a resource from where to load the definition of:

- QoS profiles
- Type definitions
- Full XML-defined DDS systems

The URI can be:

- An XML file
- A string representation of an XML document `str://"..."`
- A group of several files and strings (see `URL Groups` in the User's Manual)

The following example loads a file named "`MyProfiles.xml`" in the current directory. Then it retrieves the `DataReaderQos` from a `<qos_profile>` named "`MyProfile`" under a `<qos_library>` named "`My↩Library`" to create a `DataReader`:

```
dds::core::QosProvider my_qos_provider("MyProfiles.xml");
dds::sub::DataReader<Foo> my_reader(
        my_subscriber,
        my_topic,
        my_qos_provider.datareader_qos("MyLibrary::MyProfile"));
```

To load type definitions, see **QosProvider::type()** (p. 2044).

To create a `DomainParticipant` and its contained entities, see **QosProvider::create_participant_from_config()** (p. 2046).

8.275.2 Selecting a QoS profile

The profile name argument is optional:

```
auto reader_qos = my_qos_provider.datareader_qos();
```

When omitted, the **QosProvider** (p. 2025) looks for a profile as follows (in order of precedence):

- The profile set with **default_profile()** (p. 2041)
- The profile set in the constructor (an optional argument)
- The <qos_profile> identified as the default profile with the attribute is_default_qos=true
- If none of the above methods have been used, a QoS object with the documented default values is returned

A **QosProvider** (p. 2025) can be used in combination with functions such as **dds::pub::Publisher::default_datawriter_qos()** (p. 1998) or **dds::sub::Subscriber::default_datareader_qos()** (p. 2417) to set the profile that entities are created with by default:

```
subscriber.default_datareader_qos(
        my_qos_provider.datareader_qos("MyLibrary::MyProfile"));
// reader created with the QoS in the profile "MyLibrary::MyProfile"
dds::sub::DataReader<Foo> reader(subscriber, topic);
```

More information about QoS Profiles can be found in the QoS Profiles section, in the User's Manual.

8.275.3 The Default QosProvider

A special **QosProvider** (p. 2025), **QosProvider::Default()** (p. 2036) is always available to obtain the QoS profiles from the default locations, such as the file USER_QOS_PROFILES.xml in the current directory. For example, the following code obtains the default DataReaderQos:

```
dds::sub::qos::DataReaderQos default_reader_qos =
        dds::core::QosProvider::Default().datareader_qos();
```

This section in the User's Manual explains which profiles are automatically loaded by the default **QosProvider** (p. 2025).

The profiles that **QosProvider::Default()** (p. 2036) loads can be configured with **rti::core::default_qos_provider_params()** (p. 710).

QosProvider::Default() (p. 2036) also determines the default values for <participant_factory_qos>, which can be used to configure logging.

8.275.4 Built-in QoS Profiles

Any **QosProvider** (p. 2025), including the default **QosProvider** (p. 2025), has access to the **Builtin Qos Profiles** (p. 462) For example, to obtain the DataWriterQos from the "strict reliable" built-in profile:

```
auto strict_reliable_writer_qos =
    dds::core::QosProvider::Default().datawriter_qos(
        rti::core::builtin_profiles::qos_lib::generic_strict_reliable());
```

8.275.5 Specifying a Topic Filter

XML QoS definitions can specify a `topic_filter` attribute. Several functions (such as **datareader_qos_w_topic_name()** (p. 2039)) receive a `topic_name` argument. When these functions look for the QoS, they will try to match the `topic_name` against the `topic_filter`s, if they are used. See the `Topic Filters section` in the User's Manual.

See also

For more examples, see **Qos Provider Use Cases** (p. 604)

Examples

Foo_publisher.cxx, and **Foo_subscriber.cxx**.

8.275.6 Constructor & Destructor Documentation

8.275.6.1 QosProvider() [1/2]

```
dds::core::QosProvider::QosProvider (
            const std::string & uri,
            const std::string & profile )  [inline]
```

Create a **QosProvider** (p. 2025) fetching QoS configuration from the specified URI.

For instance, the following code:

```
dds::core::QosProvider xml_file_provider(
    "file:///somewhere/on/disk/qos-config.xml", "MyLibrary::MyProfile");
```

will create a **QosProvider** (p. 2025) which loads all of the QoS profiles from qos-config.xml and uses "MyLibrary::MyProfile" as the default profile when the profile argument to the **QosProvider** (p. 2025) getters is omitted, as described in **Selecting a QoS profile** (p. 2029).

Parameters

uri	The URI describing the location of the QoS profiles to load.
profile	The QoS profile to set as the default, overriding any is_default_qos tag in the loaded XML file.

8.275.6.2 QosProvider() [2/2]

```
dds::core::QosProvider::QosProvider (
            const std::string & uri )  [inline], [explicit]
```

Create a **QosProvider** (p. 2025) fetching QoS configuration from the specified URI.

For instance, the following code:
```
QosProvider xml_file_provider("file:///somewhere/on/disk/qos-config.xml");
```

will create a **QosProvider** (p. 2025) which loads all of the QoS profiles from qos-config.xml. When the profile argument to the **QosProvider** (p. 2025) getters is omitted, the default profile is determined as described in **Selecting a QoS profile** (p. 2029).

Parameters

uri	The URI describing the location of the QoS profiles to load. If no is_default_qos tag is found in the QoS profiles, then no profile will be set as default. Instead, the **QosProvider** (p. 2025) will be associated with the DDS default QoS values until a default profile is set using the default_library and default_profile methods.

8.275.7 Member Function Documentation

8.275.7.1 participant_qos() [1/2]

```
const dds::domain::qos::DomainParticipantQos dds::core::QosProvider::participant_qos ( ) [inline]
```

Get the **dds::domain::qos::DomainParticipantQos** (p. 1384) in the default profile.

See **Selecting a QoS profile** (p. 2029) for information on how the default profile is chosen.

Returns

The DomainParticipantQos

Examples

Foo_publisher.cxx, and **Foo_subscriber.cxx**.

8.275.7.2 participant_qos() [2/2]

```
const dds::domain::qos::DomainParticipantQos dds::core::QosProvider::participant_qos (
        const std::string & profile ) [inline]
```

Get the DomainParticipantQos from a QoS profile.

Parameters

profile	The profile from which to get the DomainParticipantQos, for example "MyLibrary::MyProfile"

Returns

The DomainParticipantQos that is located in the specified profile

8.275.7.3 topic_qos() [1/2]

```
const dds::topic::qos::TopicQos dds::core::QosProvider::topic_qos ( ) [inline]
```

Get the **dds::topic::qos::TopicQos** (p. 2509) in the default profile.

See **Selecting a QoS profile** (p. 2029) for information on how the default profile is chosen.

Returns

The TopicQos

8.275.7.4 topic_qos() [2/2]

```
const dds::topic::qos::TopicQos dds::core::QosProvider::topic_qos (
            const std::string & profile ) [inline]
```

Get the **dds::topic::qos::TopicQos** (p. 2509) from a QoS profile.

Parameters

profile	The profile from which to get the TopicQos, for example "MyLibrary::MyProfile"

Returns

The TopicQos that is located in the specified profile

8.275.7.5 subscriber_qos() [1/2]

```
const dds::sub::qos::SubscriberQos dds::core::QosProvider::subscriber_qos ( ) [inline]
```

Get the **dds::sub::qos::SubscriberQos** (p. 2423) in the default profile.

See **Selecting a QoS profile** (p. 2029) for information on how the default profile is chosen.

Returns

The SubscriberQos

8.275.7.6 subscriber_qos() [2/2]

```
const dds::sub::qos::SubscriberQos dds::core::QosProvider::subscriber_qos (
            const std::string & profile ) [inline]
```

Get the **dds::sub::qos::SubscriberQos** (p. 2423) from a QoS profile.

Parameters

profile	The profile from which to get the SubscriberQos, for example "MyLibrary::MyProfile"

Returns

The SubscriberQos that is located in the specified profile

8.275.7.7 datareader_qos() [1/2]

```
const dds::sub::qos::DataReaderQos dds::core::QosProvider::datareader_qos ( ) [inline]
```

Get the **dds::sub::qos::DataReaderQos** (p. 1072) in the default profile.

See **Selecting a QoS profile** (p. 2029) for information on how the default profile is chosen.

Note

This function doesn't take into account the `topic_filter` used in the profile, if any. See **datareader_qos_↩w_topic_name()** (p. 2039).

Returns

The DataReaderQos

Examples

Foo_subscriber.cxx.

8.275.7.8 datareader_qos() [2/2]

```
const dds::sub::qos::DataReaderQos dds::core::QosProvider::datareader_qos (
        const std::string & profile ) [inline]
```

Get the **dds::sub::qos::DataReaderQos** (p. 1072) from a QoS profile.

Note

This function doesn't take into account the `topic_filter` used in the profile, if any. See **datareader_qos_↩w_topic_name()** (p. 2039).

Parameters

profile	The profile from which to get the DataReaderQos, for example "`MyLibrary::MyProfile`"

Returns

The DataReaderQos that is located in the specified profile

8.275.7.9 publisher_qos() [1/2]

```
const dds::pub::qos::PublisherQos dds::core::QosProvider::publisher_qos ( ) [inline]
```

Get the **dds::pub::qos::PublisherQos** (p. 2007) in the default profile.

See **Selecting a QoS profile** (p. 2029) for information on how the default profile is chosen.

Returns

The PublisherQos

8.275.7.10 publisher_qos() [2/2]

```
dds::pub::qos::PublisherQos dds::core::QosProvider::publisher_qos (
        const std::string & profile ) [inline]
```

Get the **dds::pub::qos::PublisherQos** (p. 2007) from a QoS profile.

Parameters

profile	The profile from which to get the PublisherQos, for example "`MyLibrary::MyProfile`"

Returns

The PublisherQos that is located in the specified profile

8.275.7.11 datawriter_qos() [1/2]

```
const dds::pub::qos::DataWriterQos dds::core::QosProvider::datawriter_qos ( ) [inline]
```

Get the **dds::pub::qos::DataWriterQos** (p. 1226) from the default profile.

See **Selecting a QoS profile** (p. 2029) for information on how the default profile is chosen.

Note

This function doesn't take into account the `topic_filter` used in the profile, if any. See **datawriter_qos_w↩ _topic_name()** (p. 2040).

Returns

The DataWriterQos

Examples

Foo_publisher.cxx.

8.275.7.12 datawriter_qos() [2/2]

```
const dds::pub::qos::DataWriterQos dds::core::QosProvider::datawriter_qos (
        const std::string & profile ) [inline]
```

Get the **dds::pub::qos::DataWriterQos** (p. 1226) from a QoS profile.

Note

This function doesn't take into account the `topic_filter` used in the profile, if any. See **datawriter_qos_w↩ _topic_name()** (p. 2040).

Parameters

profile	The profile from which to get the DataWriterQos, for example "`MyLibrary::MyProfile`"

Returns

The DataWriterQos that is located in the specified profile

8.275.7.13 Default()

```
static QosProvider dds::core::QosProvider::Default ( ) [inline], [static]
```

Get the default **QosProvider** (p. 2025).

The default **QosProvider** (p. 2025) automatically loads XML QoS, types and application profiles from several predetermined locations, such as the file USER_QOS_PROFILES.xml in the current working directory.

Returns

The Default **QosProvider** (p. 2025)

MT Safety:

Unsafe. On VxWorks, none of the following is safe until the first **dds::domain::DomainParticipant** (p. 1326) has been created or **dds::core::QosProvider::Default()** (p. 2036) has been obtained for the first time:

- creating several **dds::domain::DomainParticipant** (p. 1326) in parallel;
- calling **dds::core::QosProvider::Default()** (p. 2036) from two or more threads in parallel;
- creating a **dds::domain::DomainParticipant** (p. 1326) and calling **dds::core::QosProvider::Default()** (p. 2036) in parallel.

See also

The Default QosProvider (p. 2029)

Default QoS (p. 1522)

Default QosProvider Use Cases (p. 605)

Examples

Foo_publisher.cxx, and **Foo_subscriber.cxx**.

8.275.7.14 reset_default()

```
static void dds::core::QosProvider::reset_default ( ) [inline], [static]
```

Reset the default settings of the default **QosProvider** (p. 2025).

If you change the default library or profile associated with the default **QosProvider** (p. 2025), calling reset_default will set the library and profile back to the default values.

8.275.7.15 default_provider_params() [1/2]

```
static  rti::core::QosProviderParams default_provider_params ( )  [static]
```

<<***extension***>> *(p. 354)* (deprecated) Get the **rti::core::QosProviderParams** (p. 2049) for the default **QosProvider** (p. 2025)

[DEPRECATED] Use **rti::core::default_qos_provider_params()** (p. 710)

Returns

rti::core::QosProviderParams (p. 2049)

8.275.7.16 default_provider_params() [2/2]

```
static void default_provider_params (
            const  rti::core::QosProviderParams & params )  [static]
```

<<***extension***>> *(p. 354)* (deprecated) Set the **rti::core::QosProviderParams** (p. 2049) for the default **QosProvider** (p. 2025)

[DEPRECATED] Use **rti::core::default_qos_provider_params()** (p. 710)

Parameters

params	The QosProviderParams to set

8.275.7.17 participant_factory_qos() [1/2]

```
dds::domain::qos::DomainParticipantFactoryQos participant_factory_qos ( ) const
```

<<***extension***>> *(p. 354)* Get the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378) in the default profile.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

See **Selecting a QoS profile** (p. 2029) for information on how the default profile is chosen.

Returns

The DomainParticipantFactoryQos

8.275.7.18 participant_factory_qos() [2/2]

```
dds::domain::qos::DomainParticipantFactoryQos participant_factory_qos (
            const std::string & profile ) const
```

<<***extension***>> *(p. 354)* Get the **dds::domain::qos::DomainParticipantFactoryQos** (p. 1378) from a QoS profile

Parameters

profile	The profile from which to get the DomainParticipantFactoryQos, for example "`MyLibrary::MyProfile`"

Returns

The DomainParticipantFactoryQos that is located in the specified profile

8.275.7.19 topic_qos_w_topic_name() [1/2]

```
const  dds::topic::qos::TopicQos topic_qos_w_topic_name (
            const std::string & profile,
            const std::string & topic_name ) const
```

<<***extension***>> *(p. 354)* Get the **dds::topic::qos::TopicQos** (p. 2509) for a topic name in a given profile.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The TopicQos will be found based on the given profile name and topic name.

Parameters

profile	The profile where the **QosProvider** (p. 2025) should look for the TopicQos
topic_name	The topic name used to select a qos within the profile, if the `topic_filter` attribute is used

Returns

dds::topic::qos::TopicQos (p. 2509) The TopicQos that is located in the specified profile

8.275.7.20 topic_qos_w_topic_name() [2/2]

```
const  dds::topic::qos::TopicQos topic_qos_w_topic_name (
            const std::string & topic_name ) const
```

<<***extension***>> *(p. 354)* Get the **dds::topic::qos::TopicQos** (p. 2509) for a topic name in the default profile.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The TopicQos will be found based on the given topic name and must be located in the default profile (see **Selecting a QoS profile** (p. 2029) for information on how the default profile is determined).

Parameters

topic_name	The topic name used to select a qos within the profile, if the `topic_filter` attribute is used

Returns

dds::topic::qos::TopicQos (p. 2509) The TopicQos that is located in the specified profile

8.275.7.21 datareader_qos_w_topic_name() [1/2]

```
const dds::sub::qos::DataReaderQos datareader_qos_w_topic_name (
            const std::string & profile,
            const std::string & topic_name ) const
```

<<***extension***>> *(p. 354)* Get the **dds::sub::qos::DataReaderQos** (p. 1072) for a topic name in a given profile.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The DataReaderQos will be found based on the given profile name and topic name.

Parameters

profile	The profile where the **QosProvider** (p. 2025) should look for the DataReaderQos
topic_name	The topic name used to select a qos within the profile, if the `topic_filter` attribute is used

Returns

dds::sub::qos::DataReaderQos (p. 1072) The DataReaderQos that is located in the specified profile

8.275.7.22 datareader_qos_w_topic_name() [2/2]

```
const dds::sub::qos::DataReaderQos datareader_qos_w_topic_name (
            const std::string & topic_name ) const
```

<<***extension***>> *(p. 354)* Get the **dds::sub::qos::DataReaderQos** (p. 1072) for a topic name in the default profile.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The DataReaderQos will be found based on the given topic name and must be located in the default profile (see **Selecting a QoS profile** (p. 2029) for information on how the default profile can is determined).

Parameters

topic_name	The topic name used to select a qos within the profile, if the `topic_filter` attribute is used

Returns

dds::sub::qos::DataReaderQos (p. 1072) The DataReaderQos that is located in the specified profile

8.275.7.23 datawriter_qos_w_topic_name() [1/2]

```
const  dds::pub::qos::DataWriterQos datawriter_qos_w_topic_name (
             const std::string & profile,
             const std::string & topic_name ) const
```

<<***extension***>> *(p. 354)* Get the **dds::pub::qos::DataWriterQos** (p. 1226) for a topic name.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The DataWriterQos will be found based on the given profile name and topic name.

Parameters

profile	The profile where the **QosProvider** (p. 2025) should look for the DataWriterQos
topic_name	The topic name used to select a qos within the profile, if the `topic_filter` attribute is used

Returns

dds::pub::qos::DataWriterQos (p. 1226) The DataWriterQos that is located in the specified profile

8.275.7.24 datawriter_qos_w_topic_name() [2/2]

```
const  dds::pub::qos::DataWriterQos datawriter_qos_w_topic_name (
             const std::string & topic_name ) const
```

<<***extension***>> *(p. 354)* Get the **dds::pub::qos::DataWriterQos** (p. 1226) for a topic name.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The DataWriterQos will be found based on the given topic name and must be located in the current default profile associated with the **QosProvider** (p. 2025).

Parameters

topic_name	The topic name used to select a qos within the profile, if the `topic_filter` attribute is used

Returns

dds::pub::qos::DataWriterQos (p. 1226) The DataWriterQos that is located in the specified profile

8.275.7.25 default_library() [1/2]

```
void default_library (
            const std::string & library_name )
```

<<***extension***>> *(p. 354)* Set the default library for this **QosProvider** (p. 2025).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

After a default library is set, the functions that expect a `profile` argument don't need the library name prefix. For example:

```
auto reader_qos = qos_provider.datareader_qos("MyLibrary::MyProfile");
// This has the same effect:
qos_provider.extensions().default_library("MyLibrary");
reader_qos = qos_provider.datareader_qos("MyProfile");
```

Parameters

library_name	The library to set as the default, or `rti::core::USE_DDS_DEFAULT_QOS_PROFILE` to unset the default library and use the system default.

8.275.7.26 default_profile() [1/2]

```
void default_profile (
            const std::string & profile_name )
```

<<***extension***>> *(p. 354)* Set the default profile for this **QosProvider** (p. 2025).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

When a default profile is set, the getters that don't receive a profile argument retrieve the QoS object from this profile (see **Selecting a QoS profile** (p. 2029)).

The profile must be located in the current default library unless the library is also specified in the name.

Parameters

profile_name	The profile to set as the default in the form `"MyLibrary::MyProfile"`, or if the **default_library()** (p. 2041) is also set, `"MyProfile"`. The special value `rti::core::USE_DDS_DEFAULT_QOS_PROFILE` unsets a previously set default profile.

8.275.7.27 default_library() [2/2]

```
rti::core::optional_value< std::string > default_library ( ) const
```

<<***extension***>> *(p. 354)* Get the default library associated with this **QosProvider** (p. 2025).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

There may not be a default library set. The default library is the library that is checked in when setting the default profile if no library name is explicitly given. The default library is configurable.

Returns

rti::core::optional_value<std::string> The default library.

8.275.7.28 default_profile() [2/2]

```
rti::core::optional_value< std::string > default_profile ( ) const
```

<<***extension***>> *(p. 354)* Get the default profile associated with this **QosProvider** (p. 2025).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

There may not be a default profile set.

Returns

rti::core::optional_value<std::string> The default profile.

8.275.7.29 default_profile_library()

```
rti::core::optional_value< std::string > default_profile_library ( ) const
```

<<***extension***>> *(*p. *354)* Get the default profile library associated with this **QosProvider** (p. 2025).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

There may not be a default profile library set. The default profile library is the library in which the default profile resides and is not necessarily the same library as the one returned from **default_library()** (p. 2041). The default profile library is not configurable.

Returns

rti::core::optional_value<std::string> The default profile library.

8.275.7.30 qos_profile_libraries()

```
dds::core::StringSeq qos_profile_libraries ( ) const
```

<<***extension***>> *(*p. *354)* Get a list of the QoS profile libraries loaded by this **QosProvider** (p. 2025).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Returns

dds::core::StringSeq (p. 436) A list of the currently loaded QoS profile libraries.

8.275.7.31 qos_profiles()

```
dds::core::StringSeq qos_profiles (
            const std::string & library_name ) const
```

<<***extension***>> *(*p. *354)* Get a list of the QoS profiles located in the default library of this **QosProvider** (p. 2025).

Returns

dds::core::StringSeq (p. 436) A list of profiles.

8.275.7.32 profiles_loaded()

```
bool profiles_loaded ( ) const
```

<<***extension***>> *(p. 354)* Check if the profiles are loaded by this **QosProvider** (p. 2025)

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Returns

bool false if the profiles are not loaded, true otherwise.

8.275.7.33 type()

```
const  dds::core::xtypes::DynamicType & type (
             const std::string & type_name ) const
```

<<***extension***>> *(p. 354)* Load a type

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Parameters

type_name	The name of the type

This function loads a type from its XML definition under the tags <dds><types>.

See also

Example (p. 610).

8.275.7.34 provider_params() [1/2]

```
rti::core::QosProviderParams provider_params ( ) const
```

<<***extension***>> *(p. 354)* Get the **QosProvider** (p. 2025) params for this **QosProvider** (p. 2025).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Returns

rti::core::QosProviderParams (p. 2049) The QosProviderParams for this **QosProvider** (p. 2025).

8.275.7.35 provider_params() [2/2]

```
void provider_params (
            const  rti::core::QosProviderParams & provider_params )
```

<<***extension***>> *(p. 354)* Set the **rti::core::QosProviderParams** (p. 2049) for this **QosProvider** (p. 2025).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Parameters

provider_params	The QosProviderParams to set.

8.275.7.36 load_profiles()

```
void load_profiles ( )
```

<<***extension***>> *(p. 354)* Load the XML QoS profiles.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The XML QoS profiles are loaded implicitly after the first **dds::domain::DomainParticipant** (p. 1326) is created or explicitly, after a call to this function. This method has the same effect as **reload_profiles()** (p. 2045).

8.275.7.37 reload_profiles()

```
void reload_profiles ( )
```

<<***extension***>> *(p. 354)* Reload the XML QoS profiles.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The XML QoS profiles are loaded implicitly after the first DomainParticipant is created or explicitly, after a call to this function. This method has the same effect as **load_profiles()** (p. 2045).

8.275.7.38 unload_profiles()

```
void unload_profiles ( )
```

<<***extension***>> *(p. 354)* Unload the XML QoS profiles.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The resources associated with the XML QoS profiles are freed. Any reference to the profiles after calling this function will fail with an **Exception** (p. 1546).

8.275.7.39 create_participant_from_config()

```
dds::domain::DomainParticipant create_participant_from_config (
            const std::string & config_name,
            const  rti::domain::DomainParticipantConfigParams & params =  rti::domain::Domain↩
ParticipantConfigParams() )
```

<<***extension***>> *(p. 354)* Creates a **dds::domain::DomainParticipant** (p. 1326) given its configuration name from a description provided in an XML configuration file that has been loaded by this **QosProvider** (p. 2025).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This operation creates a **dds::domain::DomainParticipant** (p. 1326) registering all the necessary data types and creating all the contained entities (**dds::topic::Topic** (p. 2474), **dds::pub::Publisher** (p. 1993), **dds::sub::Subscriber** (p. 2410), **dds::pub::DataWriter** (p. 1138), **dds::sub::DataReader** (p. 976)) from a description given in an XML configuration file.

The configuration name is the fully qualified name of the XML participant object, consisting of the name of the participant library plus the name of participant configuration.

For example the name "MyParticipantLibrary::PublicationParticipant" can be used to create the domain participant from the description in an XML file with contents shown in the snippet below:

```
<participant_library name="MyParticipantLibrary">

    <domain_participant name="PublicationParticipant" domain_ref="MyDomainLibrary::HelloWorldDomain">

        <publisher name="MyPublisher">

            <data_writer name="HelloWorldWriter" topic_ref="HelloWorldTopic"/>

        </publisher>

    </domain_participant>

</participant_library>
```

The entities belonging to the newly created **dds::domain::DomainParticipant** (p. 1326) can be retrieved with the help of lookup operations such as: **rti::sub::find_datareader_by_name** (p. 762).

MT Safety:

UNSAFE. On VxWorks, when enabling the RTI Security Plugins, it is not safe for multiple threads to attempt to create a **dds::domain::DomainParticipant** (p. 1326) when there are no other DomainParticipants in the Domain↩ ParticipantFactory.

See also

rti::domain::find_participant_by_name (p. 731)

dds::topic::find() (p. 694)

rti::pub::find_publisher (p. 744)

rti::sub::find_subscriber (p. 758)

rti::sub::find_datareader_by_name (p. 762)

rti::pub::find_datawriter_by_name (p. 748)

rti::pub::find_datawriter_by_name (p. 748)

rti::sub::find_datareader_by_name (p. 762)

Parameters

config_name	Name of the participant configuration in the XML file.
params	input parameters that allow changing some properties of the configuration referred to by `config_name`. Specifically, `params` allows overriding the domain ID, participant name, and entities QoS specified in the XML configuration.

See also

XML Application Creation (p. 342)

8.275.8 Friends And Related Function Documentation

8.275.8.1 USE_DDS_DEFAULT_QOS_PROFILE

```
const char* USE_DDS_DEFAULT_QOS_PROFILE  [related]
```

Special value to select the default QoS profile.

8.275.8.2 create_qos_provider_ex()

```
dds::core::detail::QosProvider create_qos_provider_ex (
            const  rti::core::QosProviderParams & params )  [related]
```

<<***extension***>> *(p. 354)* Creates a **QosProvider** (p. 2025) with parameters

Note

This is a standalone function in the namespace **rti::core** (p. 701)

A **dds::core::QosProvider** (p. 2025) is usually created with one of its constructors, but this extension function allows configuring certain parameters. By default a **QosProvider** (p. 2025) loads the profiles from certain standard locations; this function allows, for example, disabling that behavior.

8.275.8.3 default_qos_provider_params() [1/2]

```
QosProviderParams default_qos_provider_params ( )  [related]
```

<<***extension***>> *(p. 354)* Get the **rti::core::QosProviderParams** (p. 2049) for the default **QosProvider** (p. 2025)

Note

This is a standalone function in the namespace **rti::core** (p. 701)

Returns

rti::core::QosProviderParams (p. 2049)

8.275.8.4 default_qos_provider_params() [2/2]

```
void default_qos_provider_params (
            const  rti::core::QosProviderParams & params )  [related]
```

<<***extension***>> *(p. 354)* Set the **rti::core::QosProviderParams** (p. 2049) for the default **QosProvider** (p. 2025)

Note

This is a standalone function in the namespace **rti::core** (p. 701)

Use this function to configure the profiles that **dds::core::QosProvider::Default()** (p. 2036) loads.

```
rti::core::QosProviderParams params;
params.url_profile({"my_profiles.xml"});
rti::core::default_qos_provider_params(params);
auto participant_qos =
    dds::core::QosProvider::Default().participant_qos("my_library::my_profile");
```

Parameters

params	The QosProviderParams to set

8.276 rti::core::QosProviderParams Class Reference

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Configure options that control the way that XML documents containing QoS profiles are loaded by a **dds::core::QosProvider** (p. 2025).

`#include <rti/core/QosProviderParams.hpp>`

Public Member Functions

- **QosProviderParams** ()

  *Create a **QosProviderParams** (p. 2049) with default settings.*
- const **dds::core::StringSeq string_profile** () const

  *Get the current list of string profiles stored by this **QosProviderParams** (p. 2049).*
- **QosProviderParams** & **string_profile** (const **dds::core::StringSeq** &the_string_profile)

  *Set the current list of string profiles stored by this **QosProviderParams** (p. 2049).*
- const **dds::core::StringSeq url_profile** () const

  *Get the current list of **URL groups** (p. 304) stored by this **QosProviderParams** (p. 2049).*
- **QosProviderParams** & **url_profile** (const **dds::core::StringSeq** &the_url_profile)

  *Set the current list of **URL groups** (p. 304) stored by this **QosProviderParams** (p. 2049).*
- bool **ignore_user_profile** () const

  Get the value that is currently set for ignore_user_profile.
- **QosProviderParams** & **ignore_user_profile** (bool the_ignore_user_profile)

  Choose whether or not to ignore USER_QOS_PROFILES.xml.
- bool **ignore_environment_profile** () const

  Get the value that is currently set for ignore_environment_profile.
- **QosProviderParams** & **ignore_environment_profile** (bool the_ignore_environment_profile)

  *Choose whether or not to ignore the **NDDS_QOS_PROFILES** (p. 304).*
- bool **ignore_resource_profile** () const

  Get the value that is currently set for ignore_resource_profile.
- **QosProviderParams** & **ignore_resource_profile** (bool the_ignore_resource_profile)

  Choose whether or not to ignore NDDS_QOS_PROFILES.xml.

8.276.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Configure options that control the way that XML documents containing QoS profiles are loaded by a **dds::core::QosProvider** (p. 2025).

All QoS values for Entities can be configured in QoS profiles defined in XML documents. XML documents can be passed to RTI Connext in string form or, more likely, through files found on a file system.

There are also default locations where DomainParticipants will look for files to load QoS profiles. These include the current working directory from where an application is started, a file in the distribution directory for RTI Connext, and the locations specified by an environment variable.

You may disable any or all of these default locations using the Profile QoS policy.

Note: This class is equivalent to DDS_ProfileQosPolicy the other RTI Connext language APIs.

8.276.2 Constructor & Destructor Documentation

8.276.2.1 QosProviderParams()

```
rti::core::QosProviderParams::QosProviderParams ( )  [inline]
```

Create a **QosProviderParams** (p. 2049) with default settings.

8.276.3 Member Function Documentation

8.276.3.1 string_profile() [1/2]

```
const  dds::core::StringSeq rti::core::QosProviderParams::string_profile ( ) const
```

Get the current list of string profiles stored by this **QosProviderParams** (p. 2049).

Returns

dds::core::StringSeq (p. 436) The list of string profiles.

8.276.3.2 string_profile() [2/2]

```
QosProviderParams & rti::core::QosProviderParams::string_profile (
            const  dds::core::StringSeq & the_string_profile )
```

Set the current list of string profiles stored by this **QosProviderParams** (p. 2049).

Parameters

the_string_profile	A sequence of strings containing an XML document to load. The concatenation of the strings in this sequence must be a valid XML document according to the XML QoS profile schema.

[default] An empty list

8.276.3.3 url_profile() [1/2]

```
const  dds::core::StringSeq rti::core::QosProviderParams::url_profile ( ) const
```

Get the current list of **URL groups** (p. 304) stored by this **QosProviderParams** (p. 2049).

Returns

dds::core::StringSeq (p. 436) The list of string profiles.

8.276.3.4 url_profile() [2/2]

```
QosProviderParams & rti::core::QosProviderParams::url_profile (
            const  dds::core::StringSeq & the_url_profile )
```

Set the current list of **URL groups** (p. 304) stored by this **QosProviderParams** (p. 2049).

Parameters

the_url_profile	Sequence of **URL groups** (p. 304) containing a set of XML documents to load. Only one of the elements of each group will be loaded by RTI Connext, starting from the left.

[default] An empty list

8.276.3.5 ignore_user_profile() [1/2]

```
bool rti::core::QosProviderParams::ignore_user_profile ( ) const
```

Get the value that is currently set for ignore_user_profile.

Returns

bool Whether or not the **dds::core::QosProvider** (p. 2025) set with these **QosProviderParams** (p. 2049) will ignore the USER_QOS_PROFILES.xml.

8.276.3.6 ignore_user_profile() [2/2]

```
QosProviderParams & rti::core::QosProviderParams::ignore_user_profile (
            bool the_ignore_user_profile )
```

Choose whether or not to ignore USER_QOS_PROFILES.xml.

When this field is set to true, the QoS profiles contained in the file USER_QOS_PROFILES.xml in the current working directory will not be loaded.

Parameters

the_ignore_user_profile	Whether or not to ignore the USER_QOS_PROFILES.xml in the current working directory.

Returns

QosProviderParams (p. 2049)& This **QosProviderParams** (p. 2049)

[default] false

8.276.3.7 ignore_environment_profile() [1/2]

```
bool rti::core::QosProviderParams::ignore_environment_profile ( ) const
```

Get the value that is currently set for ignore_environment_profile.

Returns

bool Whether or not the **dds::core::QosProvider** (p. 2025) set with these **QosProviderParams** (p. 2049) will ignore the **NDDS_QOS_PROFILES** (p. 304).

8.276.3.8 ignore_environment_profile() [2/2]

```
QosProviderParams & rti::core::QosProviderParams::ignore_environment_profile (
            bool the_ignore_environment_profile )
```

Choose whether or not to ignore the **NDDS_QOS_PROFILES** (p. 304).

When this field is set to true, the QoS profiles pointed to by the **NDDS_QOS_PROFILES** (p. 304) will not be loaded.

Parameters

the_ignore_environment_profile	Whether or not to ignore **NDDS_QOS_PROFILES** (p. 304).

Returns

QosProviderParams (p. 2049)& This **QosProviderParams** (p. 2049)

[default] false

8.276.3.9 ignore_resource_profile() [1/2]

```
bool rti::core::QosProviderParams::ignore_resource_profile ( ) const
```

Get the value that is currently set for ignore_resource_profile.

Returns

bool Whether or not the **dds::core::QosProvider** (p. 2025) set with these **QosProviderParams** (p. 2049) will ignore NDDS_QOS_PROFILES.xml.

8.276.3.10 ignore_resource_profile() [2/2]

```
QosProviderParams & rti::core::QosProviderParams::ignore_resource_profile (
            bool the_ignore_resource_profile )
```

Choose whether or not to ignore NDDS_QOS_PROFILES.xml.

When this field is set to true, the QoS profiles contained in the file NDDS_QOS_PROFILES.xml in $↩ NDDSHOME/resource/qos_profiles_../xml will not be loaded.

Parameters

the_ignore_resource_profile	Whether or not to ignore the NDDS_QOS_PROFILES.xml.

Returns

QosProviderParams (p. 2049)& This **QosProviderParams** (p. 2049)

[default] false

8.277 dds::sub::Query Class Reference

<<***value-type***>> *(p. 350)* Encapsulates a query for a **dds::sub::cond::QueryCondition** (p. 2059).

```
#include <dds/sub/Query.hpp>
```

Public Member Functions

- template<typename T >
  Query (const **dds::sub::DataReader**< T > &reader, const std::string &query_expression)

  Creates a query.

- template<typename T , typename FWIterator >
  Query (const **dds::sub::DataReader**< T > &reader, const std::string &query_expression, const FWIterator &params_begin, const FWIterator &params_end)

  Creates a query with the expression parameters in an iterator range.
- template<typename T >
  Query (const **dds::sub::DataReader**< T > &reader, const std::string &query_expression, const std::vector< std::string > &params)

  Creates a query with the expression parameters in a vector.
- const std::string & **expression** () const

  Gets the expression.
- const_iterator **begin** () const

  Provides the begin iterator to the parameter list.
- const_iterator **end** () const

  Provides the end iterator to the parameter list.
- iterator **begin** ()

  Provides the begin iterator to the parameter list.
- iterator **end** ()

  Provides the end iterator to the parameter list.
- template<typename FWIterator >
  void **parameters** (const FWIterator &the_begin, const FWIterator the_end)

  Sets the parameters for the expression.
- void **add_parameter** (const std::string &param)

  Appends a parameter.
- size_t **parameters_length** () const

  Gets the number of parameters.
- const std::vector< std::string > & **parameters** () const

  <<***extension***>> *(*p. *354) Gets the parameters*
- std::string **name** () const

  <<***extension***>> *(*p. *354) Gets the filter name*
- **dds::sub::Query** & **name** (const std::string &the_name)

  <<***extension***>> *(*p. *354) Sets a filter name*
- const **AnyDataReader** & **data_reader** () const

  *Gets the related **DataReader** (*p. *976).*

8.277.1 Detailed Description

<<***value-type***>> *(*p. *350)* Encapsulates a query for a **dds::sub::cond::QueryCondition** (p. 2059).

A query contains the **DataReader** (p. 976) to query data from, an expression and optionally a list of parameters.

Queries and Filters Syntax (p. 84) defines the syntax of the expression and the parameters.

See also

Filtering with Query Conditions (p. 332)

8.277.2 Constructor & Destructor Documentation

8.277.2.1 Query() [1/3]

```
template<typename T >
dds::sub::Query::Query (
            const  dds::sub::DataReader< T > & reader,
            const std::string & query_expression )  [inline]
```

Creates a query.

Template Parameters

T	The topic-type of the **DataReader** (p. 976)

Parameters

reader	**DataReader** (p. 976) to query data from
query_expression	Expression describing the query

8.277.2.2 Query() [2/3]

```
template<typename T , typename FWIterator >
dds::sub::Query::Query (
            const  dds::sub::DataReader< T > & reader,
            const std::string & query_expression,
            const FWIterator & params_begin,
            const FWIterator & params_end )  [inline]
```

Creates a query with the expression parameters in an iterator range.

Template Parameters

T	The topic-type of the **DataReader** (p. 976)
FWIterator	A forward iterator whose value type is std::string (or convertible to std::string)

Parameters

reader	**DataReader** (p. 976) to query data from
query_expression	Expression describing the query
params_begin	The beginning of a range of parameters for the query expression

Parameters

params_end	The end of the range

8.277.2.3 Query() [3/3]

```
template<typename T >
dds::sub::Query::Query (
             const  dds::sub::DataReader< T > & reader,
             const std::string & query_expression,
             const std::vector< std::string > & params )  [inline]
```

Creates a query with the expression parameters in a vector.

Template Parameters

T	The topic-type of the **DataReader** (p. 976)

Parameters

reader	**DataReader** (p. 976) to query data from
query_expression	Expression describing the query
params	The parameters for the query expression

8.277.3 Member Function Documentation

8.277.3.1 expression()

```
const std::string & dds::sub::Query::expression ( ) const  [inline]
```

Gets the expression.

8.277.3.2 begin() [1/2]

```
const_iterator dds::sub::Query::begin ( ) const  [inline]
```

Provides the begin iterator to the parameter list.

Returns

A random-access iterator whose value type is std::string

8.277.3.3 end() [1/2]

```
const_iterator dds::sub::Query::end ( ) const  [inline]
```

Provides the end iterator to the parameter list.

Returns

A random-access iterator whose value type is std::string

8.277.3.4 begin() [2/2]

```
iterator dds::sub::Query::begin ( )  [inline]
```

Provides the begin iterator to the parameter list.

Returns

A random-access iterator whose value type is std::string

8.277.3.5 end() [2/2]

```
iterator dds::sub::Query::end ( )  [inline]
```

Provides the end iterator to the parameter list.

Returns

A random-access iterator whose value type is std::string

8.277.3.6 parameters() [1/2]

```
template<typename FWIterator >
void dds::sub::Query::parameters (
            const FWIterator & the_begin,
            const FWIterator the_end )  [inline]
```

Sets the parameters for the expression.

Template Parameters

FWIterator	A forward iterator whose value type is std::string (or convertible to std::string)

Parameters

the_begin	The beginning of a range of parameters for the query expression
the_end	The end of the range

8.277.3.7 add_parameter()

```
void dds::sub::Query::add_parameter (
            const std::string & param )  [inline]
```

Appends a parameter.

8.277.3.8 parameters_length()

```
size_t dds::sub::Query::parameters_length ( ) const  [inline]
```

Gets the number of parameters.

8.277.3.9 parameters() [2/2]

```
const std::vector< std::string > & dds::sub::Query::parameters ( ) const
```

<<***extension***>> *(*p. *354)* Gets the parameters

8.277.3.10 name() [1/2]

```
std::string dds::sub::Query::name ( ) const
```

<<***extension***>> *(*p. *354)* Gets the filter name

Returns

The name of the filter, or an empty string when using the default SQL filter.

8.277.3.11 name() [2/2]

```
dds::sub::Query & dds::sub::Query::name (
            const std::string & the_name )
```

<<***extension***>> *(p. 354)* Sets a filter name

You can use one of the built-in filters (**rti::topic::sql_filter_name** (p. 49) and **rti::topic::stringmatch_filter_name** (p. 49)), or a custom filter, which you need to register using **dds::domain::DomainParticipant::register_contentfilter()** (p. 1349).

[default] Empty string (equivalent to **rti::topic::sql_filter_name** (p. 49))

8.277.3.12 data_reader()

```
const AnyDataReader & dds::sub::Query::data_reader ( ) const [inline]
```

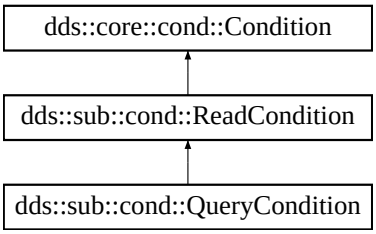
Gets the related **DataReader** (p. 976).

8.278 dds::sub::cond::QueryCondition Class Reference

<<***reference-type***>> *(p. 351)* Specialized **ReadCondition** (p. 2133) that allows applications to also specify a filter on the data available in a **dds::sub::DataReader** (p. 976)

```
#include <dds/sub/cond/QueryCondition.hpp>
```

Inheritance diagram for dds::sub::cond::QueryCondition:



Public Types

- typedef DELEGATE::iterator **iterator**

  Parameter iterator. A random-access iterator of std::string.
- typedef DELEGATE::const_iterator **const_iterator**

  Parameter iterator. A random-access iterator of std::string.

Public Member Functions

- **QueryCondition** (const **dds::sub::Query** &query, const **dds::sub::status::DataState** &status)

  *Creates a **QueryCondition** (p. 2059).*
- template<typename Functor >
  QueryCondition (const **dds::sub::Query** &query, const **dds::sub::status::DataState** &status, const Functor &functor)

  *Creates a **QueryCondition** (p. 2059) with a handler.*
- std::string **expression** () const

  Gets the expression.
- **dds::core::StringSeq parameters** () const

  Gets the parameters.
- template<typename FWIterator >
  void **parameters** (const FWIterator & **begin**, const FWIterator & **end**)

  Modifies the query parameters.
- uint32_t **parameters_length** () const

  Gets the number of parameters.
- void **parameters** (const **dds::core::StringSeq** &parameters)

  <<***extension***>> *(p. 354) Set the parameters*

Additional Inherited Members

8.278.1 Detailed Description

<<***reference-type***>> *(p. 351)* Specialized **ReadCondition** (p. 2133) that allows applications to also specify a filter on the data available in a **dds::sub::DataReader** (p. 976)

Each query condition filter is composed of a **dds::sub::cond::ReadCondition** (p. 2133) state filter and a content filter expressed as a `query_expression` and `query_parameters`.

The query (`query_expression`) is similar to an SQL WHERE clause and can be parameterised by arguments that are dynamically changeable by the set_query_parameters() operation.

Two query conditions that have the same `query_expression` will require unique query condition content filters if their `query_paramters` differ. **Query** (p. 2053) conditions that differ only in their state masks will share the same query condition content filter.

Queries and Filters Syntax (p. 84) describes the syntax of `query_expression` and `query_parameters`.

See also

Filtering with Query Conditions (p. 332)

8.278.2 Member Typedef Documentation

8.278.2.1 iterator

```
typedef DELEGATE::iterator dds::sub::cond::QueryCondition::iterator
```

Parameter iterator. A random-access iterator of std::string.

8.278.2.2 const_iterator

```
typedef DELEGATE::const_iterator dds::sub::cond::QueryCondition::const_iterator
```

Parameter iterator. A random-access iterator of std::string.

8.278.3 Constructor & Destructor Documentation

8.278.3.1 QueryCondition() [1/2]

```
dds::sub::cond::QueryCondition::QueryCondition (
            const  dds::sub::Query & query,
            const  dds::sub::status::DataState & status )  [inline]
```

Creates a **QueryCondition** (p. 2059).

Parameters

query	The query
status	The sample, view and instance states of interest.

See also

ReadCondition (p. 2133)

rti::sub::cond::create_query_condition_ex

8.278.3.2 QueryCondition() [2/2]

```
template<typename Functor >
dds::sub::cond::QueryCondition::QueryCondition (
            const  dds::sub::Query & query,
```

```
            const  dds::sub::status::DataState & status,
            const Functor & functor )  [inline]
```

Creates a **QueryCondition** (p. 2059) with a handler.

This constructor is similar to **this ReadCondition constructor** (p. 2134).

See also

Filtering with Query Conditions (p. 332) for an example on how to use this constructor.

8.278.4 Member Function Documentation

8.278.4.1 expression()

```
std::string dds::sub::cond::QueryCondition::expression ( ) const  [inline]
```

Gets the expression.

8.278.4.2 parameters() [1/3]

```
 dds::core::StringSeq dds::sub::cond::QueryCondition::parameters ( ) const  [inline]
```

Gets the parameters.

8.278.4.3 parameters() [2/3]

```
template<typename FWIterator >
void dds::sub::cond::QueryCondition::parameters (
            const FWIterator & begin,
            const FWIterator & end )  [inline]
```

Modifies the query parameters.

Template Parameters

FWIterator	A forward iterator whose value type is std::string (or convertible to std::string).

Parameters

begin	The beginning of the range of parameters
end	The end of the range

8.278.4.4 parameters_length()

```
uint32_t dds::sub::cond::QueryCondition::parameters_length ( ) const  [inline]
```

Gets the number of parameters.

8.278.4.5 parameters() [3/3]

```
void parameters (
            const  dds::core::StringSeq & parameters )
```

<<***extension***>> *(*p. *354)* Set the parameters

Note

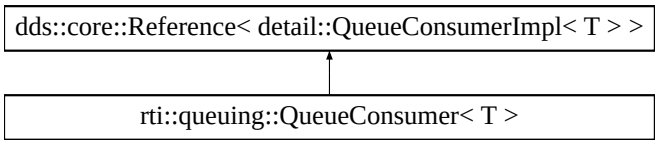
This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.279 rti::queuing::QueueConsumer< T > Class Template Reference

Allows you to receive samples from a SharedReaderQueue.

```
#include <rti/queuing/QueueConsumer.hpp>
```

Inheritance diagram for rti::queuing::QueueConsumer< T >:



Public Member Functions

- **QueueConsumer** (const **QueueConsumerParams** &params, bool is_enabled=true, **Listener** *consumer_listener=NULL, const **rti::core::Guid** &consumer_guid= **rti::core::Guid::unknown**())

  Creates a queue consumer.
- **QueueConsumer** (const **QueueConsumerParams** &params, bool is_enabled, std::shared_ptr< **Listener** > consumer_listener, const **rti::core::Guid** &consumer_guid= **rti::core::Guid::unknown**())

  *Creates a **QueueConsumer** (p. 2063) with parameters.*
- void **listener** (**Listener** *the_listener, const **dds::core::status::StatusMask** &event_mask= **dds::core::status::StatusMask::none**())

  Sets the listener associated with this consumer.
- **Listener** * **listener** () const

  Returns the listener currently associated with this Consumer.
- std::shared_ptr< **Listener** > **get_listener** () const

  Gets the listener.
- void **set_listener** (std::shared_ptr< **Listener** > the_listener)

  Sets a listener to be notified of status updates.
- void **enable** ()

  *Enables the **QueueConsumer** (p. 2063) to receive data and listener notifications.*
- void **acknowledge_sample** (const **dds::sub::SampleInfo** &sample_info, bool is_positive_acknowledgment=true)

  Explicitly acknowledges a single sample.
- void **acknowledge_all** (bool is_positive_acknowledgment=true)

  Acknowledge all previously accessed samples.
- **dds::sub::DataReader**< T > **reader** () const

  *Retrieves the underlying **dds::sub::DataReader** (p. 976).*
- **rti::core::Guid** **guid** () const

  *Returns the GUID of this **QueueConsumer** (p. 2063).*
- **dds::sub::LoanedSamples**< T > **receive_samples** (const **dds::core::Duration** &max_wait)

  Waits for multiple samples and provides a loaned container to access them.
- **dds::sub::LoanedSamples**< T > **receive_samples** (int min_count, int max_count, const **dds::core::Duration** &max_wait)

  Waits for multiple samples and provides a loaned container to access them.
- **dds::sub::LoanedSamples**< T > **take_samples** ()

  Provides a loaned container to access the existing samples.
- **dds::sub::LoanedSamples**< T > **take_samples** (int max_count)

  Provides a loaned container to access the existing samples.
- **dds::sub::LoanedSamples**< T > **read_samples** ()

  Provides a loaned container to access the existing samples.
- **dds::sub::LoanedSamples**< T > **read_samples** (int max_count)

  Provides a loaned container to access the existing samples.
- bool **wait_for_samples** (const **dds::core::Duration** &max_wait)

  Waits for samples.
- bool **wait_for_samples** (int min_count, const **dds::core::Duration** &max_wait)

  Waits for samples.
- void **send_availability** (**ConsumerAvailabilityParams** parameters)

  *Sends the **QueueConsumer** (p. 2063) availability to Queuing Service.*
- bool **has_matching_reader_queue** ()

  *Checks whether this **QueueConsumer** (p. 2063) has matched at least one SharedReaderQueue.*

8.279.1 Detailed Description

template<typename T>
class rti::queuing::QueueConsumer< T >

Allows you to receive samples from a SharedReaderQueue.

A **QueueConsumer** (p. 2063) is an entity that allows you to receive samples from a SharedReaderQueue hosted by Queuing Service. A **QueueConsumer** (p. 2063) has one underlying **dds::sub::DataReader** (p. 976) to communicate with a SharedReaderQueue.

Valid types for the topic of the DataReader (T) are those generated by rtiddsgen, the **DDS built-in types** (p. 50), and **dds::core::xtypes::DynamicData** (p. 1461).

To receive samples from a SharedReaderQueue, a **QueueConsumer** (p. 2063) must set its topic name (see Queue↩ ConsumerParams::queue_topic_name) equal to the SharedReaderQueue topic name (set with the XML tag <topic_↩ name> under <shared_reader_queue>).

In addition, the **QueueConsumer** (p. 2063) must set its SharedSubscriber name (QueueConsumerParams::shared_↩ subscriber_name) equal to the name of the SharedSubscriber hosting the SharedReaderQuueue.

The **QueueConsumer** (p. 2063) and the SharedReaderQueue must also be in the same DDS domain (that is, they must have the same domain ID).

A **QueueConsumer** (p. 2063) has an associated **dds::domain::DomainParticipant** (p. 1326), which can be shared with other QueueConsumers or RTI Connext routines. All the other DDS entities required for queuing interaction, including the **dds::sub::DataReader** (p. 976) to receive samples, are automatically created when the **QueueConsumer** (p. 2063) is constructed.

Quality of Service (QoS) for the underlying DataReader is configurable (see **QueueConsumerParams::qos_profile** (p. 2079) and QueueConsumerParams::datareader_qos).

If no QoS is specified in the **QueueConsumerParams** (p. 2077), the default DataReaderQos will be used.

A **QueueConsumer** (p. 2063) must always be reliable in order to match and communicate with the SharedReader↩ Queue. In addition, it shall use application acknowledgement to notify Queuing Service that a sample has been consumed successfully. Hence, whichever DataReaderQos is selected, the following QoS settings are always overriden:

- **dds::core::policy::Reliability::kind** (p. 2153) is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157).
- **dds::core::policy::Reliability::acknowledgment_kind** (p. 2154) is set to **rti::core::policy::Acknowledgment↩ Kind_def::APPLICATION_EXPLICIT** (p. 798).
- **rti::core::policy::DataReaderResourceLimits::max_app_ack_response_length** (p. 1101) is set to 1.

The underlying DataReader is created with an internal **dds::sub::DataReaderListener** (p. 1055), which is required to provide the behavior of a **QueueConsumer** (p. 2063), including notification of events via **QueueConsumerListener** (p. 2075). For this reason, when accessing the underlying DataReader, the listener should not be set, removed or modified; otherwise the behavior of the **QueueConsumer** (p. 2063) will be incorrect and unpredictable.

Optionally, a **QueueConsumer** (p. 2063) can be configured to create an availability channel (QueueConsumerParams↩ ::enable_availability). This channel is used to report the **QueueConsumer** (p. 2063)'s availability to the SharedReader↩ Queue from which the **QueueConsumer** (p. 2063) receives samples. Queuing Service uses the reported availability to distribute samples to the **QueueConsumer** (p. 2063).

If the availability channel is enabled for the **QueueConsumer** (p. 2063), a **dds::pub::DataWriter** (p. 1138) is created to send availability samples to the SharedReaderQueue.

You can configure the DataWriterQos used by the availability DataWriter by using **QueueConsumerParams::qos_↩ profile** (p. 2079).

Whichever DataWriterQos is selected, the following QoS settings are always overriden:

- **dds::core::policy::Reliability::kind** (p. 2153) is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157).
- **dds::core::policy::Durability::kind** (p. 1437) is set to **dds::core::policy::DurabilityKind_def::TRANSIENT_↩ LOCAL** (p. 1442).
- **dds::core::policy::History::kind** (p. 1604) is set to **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607).

Template Parameters

T	The data type for the SharedReaderQueue topic

See also

QueueConsumer (p. 2063)

QueueConsumerListener (p. 2075)

8.279.2 Constructor & Destructor Documentation

8.279.2.1 QueueConsumer() [1/2]

```
template<typename T >
rti::queuing::QueueConsumer< T >::QueueConsumer (
             const  QueueConsumerParams & params,
             bool is_enabled = true,
              Listener * consumer_listener = NULL,
```

```
            const  rti::core::Guid & consumer_guid =  rti::core::Guid::unknown() )  [inline],
[explicit]
```

Creates a queue consumer.

[DEPRECATED] When using a listener, prefer the constructor that receives a shared_ptr<**Listener** (p. 1637)> instead of a regular Listener* pointer.

8.279.2.2 QueueConsumer() [2/2]

```
template<typename T >
rti::queuing::QueueConsumer< T >::QueueConsumer (
            const  QueueConsumerParams & params,
            bool is_enabled,
            std::shared_ptr<  Listener > consumer_listener,
            const  rti::core::Guid & consumer_guid =  rti::core::Guid::unknown() )  [inline],
[explicit]
```

Creates a **QueueConsumer** (p. 2063) with parameters.

Parameters

params	All the parameters that configure this **QueueConsumer** (p. 2063)
is_enabled	Specifies if the **QueueConsumer** (p. 2063) is created ready to receive notifications in its listener and receive data. If you choose to bind the **QueueConsumer** (p. 2063) to a **QueueConsumerListener** (p. 2075) using an **rti::core::ListenerBinder** (p. 1641) you should create the **QueueConsumer** (p. 2063) disabled to avoid missing listener notifications. You can later enable the disabled **QueueConsumer** (p. 2063) using **QueueConsumer::enable** (p. 2069).
consumer_listener	A shared_ptr to a **QueueConsumerListener** (p. 2075) object to receive event notifications.
consumer_guid	A GUID identifier for the consumer. When consumer_guid is set to **rti::core::Guid::unknown()** (p. 1597) the Guid identifier is generated automatically. You can retrieve the generated **QueueConsumer** (p. 2063) Guid using the **QueueConsumer::guid** (p. 2070) method. To restart a **QueueConsumer** (p. 2063) just create a new one with its same GUID.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

QueueConsumerParams (p. 2077)

8.279.3 Member Function Documentation

8.279.3.1 listener() [1/2]

```
template<typename T >
void  rti::queuing::QueueConsumer< T >::listener (
              Listener * the_listener,
             const  dds::core::status::StatusMask & event_mask =  dds::core::status::StatusMask↩
::none() )  [inline]
```

Sets the listener associated with this consumer.

[DEPRECATED] The use of **set_listener()** (p. 2068) is recommended. Unlike this function, set_listener receives a shared_ptr which simplifies the management of listener's lifecycle.

Parameters

the_listener	The **QueueConsumerListener** (p. 2075) to set
event_mask	The **dds::core::status::StatusMask** (p. 2375) associated with the listener

8.279.3.2 listener() [2/2]

```
template<typename T >
Listener *  rti::queuing::QueueConsumer< T >::listener ( ) const  [inline]
```

Returns the listener currently associated with this Consumer.

[DEPRECATED] Prefer **get_listener()** (p. 2068) instead of this function.

If there is no listener it returns NULL.

8.279.3.3 get_listener()

```
template<typename T >
std::shared_ptr<  Listener >  rti::queuing::QueueConsumer< T >::get_listener ( ) const  [inline]
```

Gets the listener.

8.279.3.4 set_listener()

```
template<typename T >
void  rti::queuing::QueueConsumer< T >::set_listener (
             std::shared_ptr<  Listener > the_listener )  [inline]
```

Sets a listener to be notified of status updates.

Warning

It's recommended that the listener implementation doesn't hold a permanent reference to this object. If it does, the application needs to manually reset the listener or manually close this object to ensure that there is no cycle that prevents the destruction of these two objects.

Parameters

the_listener	A shared pointer to the listener to receive updates or `nullptr` to reset the current listener and stop receiving updates.

8.279.3.5 enable()

```
template<typename T >
void  rti::queuing::QueueConsumer< T >::enable ( )  [inline]
```

Enables the **QueueConsumer** (p. 2063) to receive data and listener notifications.

If you create the **QueueConsumer** (p. 2063) disabled you can enable it using this mehtod. To avoid missing listener notifciation when you use an **rti::core::ListenerBinder** (p. 1641) to bind the **QueueConsumer** (p. 2063) to a **Queue↩ ConsumerListener** (p. 2075) use **QueueConsumer::enable** (p. 2069) to enable the **QueueConsumer** (p. 2063) after it is bound to the listener.

8.279.3.6 acknowledge_sample()

```
template<typename T >
void  rti::queuing::QueueConsumer< T >::acknowledge_sample (
            const  dds::sub::SampleInfo & sample_info,
            bool is_positive_acknowledgment = true )  [inline]
```

Explicitly acknowledges a single sample.

This operation calls **dds::sub::DataReader::acknowledge_sample** (p. 1015) on the underlying DataReader.

The parameter is_positive_acknowledgment indicates whether or not the acknowledged sample was successfully processed by the application. If the sample was not processed successfully, Queuing Service will try to redeliver it to a different **QueueConsumer** (p. 2063) according to the delivery policy of the SharedReaderQueue.

Parameters

sample_info	Identifies the sample being acknowledged. The sampleInfo can be extracted from Sample::info.
is_positive_acknowledgment	Indicates whether or not the sample was succesfully processed.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.279.3.7 acknowledge_all()

```
template<typename T >
void  rti::queuing::QueueConsumer< T >::acknowledge_all (
            bool is_positive_acknowledgment = true )  [inline]
```

Acknowledge all previously accessed samples.

This is equivalent to calling **QueueConsumer::acknowledge_sample(const dds::sub::SampleInfo&, bool)** (p. 2069) for every single previously accessed sample.

Parameters

is_positive_acknowledgment	Indicates whether or not the acknowledged samples were successfully processed by the application. If the samples were not processed successfully, Queuing Service will try to redeliver them to a different **QueueConsumer** (p. 2063) according to the delivery policy of the SharedReaderQueue.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

QueueConsumer::acknowledge_sample(const dds::sub::SampleInfo&, bool) (p. 2069)

8.279.3.8 reader()

```
template<typename T >
dds::sub::DataReader< T >  rti::queuing::QueueConsumer< T >::reader ( ) const  [inline]
```

Retrieves the underlying **dds::sub::DataReader** (p. 976).

Accessing the DataReader may be useful for a number of advanced use cases, such as getting DataReader protocol or cache statuses.

MT Safety:

SAFE

See also

dds::sub::DataReader (p. 976)

dds::sub::DataReader (p. 976)

dds::sub::DataReader::datareader_protocol_status (p. 1013)

8.279.3.9 guid()

```
template<typename T >
rti::core::Guid  rti::queuing::QueueConsumer< T >::guid ( ) const  [inline]
```

Returns the GUID of this **QueueConsumer** (p. 2063).

The GUID of the **QueueConsumer** (p. 2063) is determined based on the value of QueueConsumerParams::entity_↩ name and QueueConsumerParams::queue_topic_name. Note that the **QueueConsumer** (p. 2063) GUID may be equal or different than the GUID of the underlying DataReader virtual GUID.

The GUID identifies a **QueueConsumer** (p. 2063) and the samples it consumes. The underlying DataReader's content filter on the related_reader_guid is set to the **QueueConsumer** (p. 2063)'s GUID.

See also

QueueConsumerParams::entity_name

QueueProducer::guid (p. 2089)

8.279.3.10 receive_samples() [1/2]

```
template<typename T >
dds::sub::LoanedSamples< T >  rti::queuing::QueueConsumer< T >::receive_samples (
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for multiple samples and provides a loaned container to access them.

Equivalent to using **QueueConsumer::receive_samples(int, int, const dds::core::Duration&)** (p. 2071) with `min↩ _count=1` and `max_count=`**`dds::core::LENGTH_UNLIMITED`** (p. 438).

See also

dds::sub::LoanedSamples (p. 1663)

QueueConsumer::receive_samples(int, int, const dds::core::Duration&) (p. 2071)

8.279.3.11 receive_samples() [2/2]

```
template<typename T >
dds::sub::LoanedSamples< T >  rti::queuing::QueueConsumer< T >::receive_samples (
            int min_count,
            int max_count,
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for multiple samples and provides a loaned container to access them.

Equivalent to using **QueueConsumer::wait_for_samples(int, const dds::core::Duration&)** (p. 2073) and **Queue↩ Consumer::take_samples(int)** (p. 2072).

See also

dds::sub::LoanedSamples (p. 1663)

QueueConsumer::wait_for_samples(int, const dds::core::Duration&) (p. 2073)

QueueConsumer::take_samples(int) (p. 2072)

8.279.3.12 take_samples() [1/2]

```
template<typename T >
dds::sub::LoanedSamples< T >  rti::queuing::QueueConsumer< T >::take_samples ( )  [inline]
```

Provides a loaned container to access the existing samples.

This operation is equivalent to using **QueueConsumer::take_samples(int)** (p. 2072) with `max_count=`**dds::core::LENGTH_UNLIMITED** (p. 438).

See also

dds::sub::LoanedSamples (p. 1663)

QueueConsumer::take_samples(int) (p. 2072)

8.279.3.13 take_samples() [2/2]

```
template<typename T >
dds::sub::LoanedSamples< T >  rti::queuing::QueueConsumer< T >::take_samples (
            int max_count )  [inline]
```

Provides a loaned container to access the existing samples.

Takes all the existing samples up to `max_count` and provides a loaned container to access them.

This operation does not make a copy of the data.

The loan is returned with **dds::sub::LoanedSamples::return_loan** (p. 1667).

This operation may be used after a call to **QueueConsumer::wait_for_samples(int, const dds::core::Duration&)** (p. 2073).

Parameters

max_count	The maximum number of samples that are taken at a time. The special value **dds::core::LENGTH_UNLIMITED** (p. 438) may be used. This value will read up to the limit specified by **rti::core::policy::DataReaderResourceLimits::max_samples_per_read** (p. 1091).

Returns

A container with up to max_count elements. May be empty if there were no replies to get.

Exceptions

One	of the **Standard Exceptions** (p. 428)

MT Safety:

SAFE

See also

Sample

dds::sub::LoanedSamples (p. 1663)

dds::sub::LoanedSamples::return_loan (p. 1667)

QueueConsumer::wait_for_samples(int, const dds::core::Duration&) (p. 2073)

dds::sub::DataReader::take (p. 997) (for a more detailed description on how QoS and other **parameters** (p. 2063) affect the underlying DataReader)

QueueConsumer::take_samples() (p. 2071)

8.279.3.14 read_samples() [1/2]

```
template<typename T >
dds::sub::LoanedSamples< T >  rti::queuing::QueueConsumer< T >::read_samples ( )  [inline]
```

Provides a loaned container to access the existing samples.

This operation is equivalent to **QueueConsumer::take_samples()** (p. 2071), except the samples remain in the **QueueConsumer** (p. 2063) and can be read or taken again.

8.279.3.15 read_samples() [2/2]

```
template<typename T >
dds::sub::LoanedSamples< T >  rti::queuing::QueueConsumer< T >::read_samples (
            int max_count )  [inline]
```

Provides a loaned container to access the existing samples.

This operation is equivalent to **QueueConsumer::take_samples(int)** (p. 2072), except the samples remain in the **QueueConsumer** (p. 2063) and can be read or taken again.

8.279.3.16 wait_for_samples() [1/2]

```
template<typename T >
bool  rti::queuing::QueueConsumer< T >::wait_for_samples (
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for samples.

This operation is equivalent to **QueueConsumer::wait_for_samples(int, const dds::core::Duration&)** (p. 2073) with `min_count=1`.

See also

QueueConsumer::wait_for_samples(int, const dds::core::Duration&) (p. 2073)

8.279.3.17 wait_for_samples() [2/2]

```
template<typename T >
bool  rti::queuing::QueueConsumer< T >::wait_for_samples (
            int min_count,
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for samples.

This operation waits for min_count samples to be available. It will wait up to max_wait .

If this operation is called several times but the available samples are not taken (with **QueueConsumer::take_samples(int)** (p. 2072)), this operation may return immediately and will not wait for new samples. New samples may replace existing ones if they are not taken, depending on the **dds::core::policy::History** (p. 1602) used to configure this **QueueConsumer** (p. 2063).

Parameters

min_count	Minimum number of samples that need to be available for this operation to unblock.
max_wait	Maximum waiting time after which this operation unblocks regardless of how many samples are available.

Returns

true if at least min_count samples were available before max_wait elapsed, or false otherwise.

MT Safety:

Concurrent calls to this operation on the same object are not allowed.

See also

QueueConsumer::take_samples(int) (p. 2072)

8.279.3.18 send_availability()

```
template<typename T >
void  rti::queuing::QueueConsumer< T >::send_availability (
             ConsumerAvailabilityParams parameters )  [inline]
```

Sends the **QueueConsumer** (p. 2063) availability to Queuing Service.

This method will use the availability DataWriter to send a sample to Queue Service indicating the availability status of the **QueueConsumer** (p. 2063) application.

The method throws **dds::core::PreconditionNotMetError** (p. 1941) if availability is not enabled for this **QueueConsumer** (p. 2063) using QueueConsumerParams::enable_availability.

Exceptions

One	of the **Standard Exceptions** (p. 428) ;

Parameters

parameters	Availability status

MT Safety:

SAFE

See also

QueueConsumerParams::enable_availability

8.279.3.19 has_matching_reader_queue()

```
template<typename T >
bool  rti::queuing::QueueConsumer< T >::has_matching_reader_queue ( )  [inline]
```

Checks whether this **QueueConsumer** (p. 2063) has matched at least one SharedReaderQueue.

Exceptions

One	of the **Standard Exceptions** (p. 428) ;

Returns

True if this **QueueConsumer** (p. 2063) matches at least one SharedReaderQueue.

MT Safety:

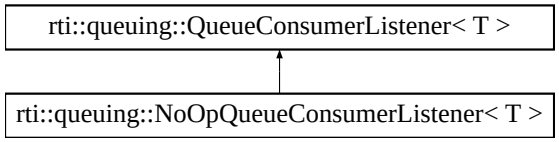
SAFE

8.280 rti::queuing::QueueConsumerListener< T > Class Template Reference

Called when certain events occur in a **QueueConsumer** (p. 2063).

```
#include <rti/queuing/QueueConsumerListener.hpp>
```

Inheritance diagram for rti::queuing::QueueConsumerListener< T >:



Public Member Functions

- virtual void **on_sample_available** (**QueueConsumer**< T > &consumer)=0

  User callback.
- virtual void **on_shared_reader_queue_matched** (**QueueConsumer**< T > &consumer, const **dds::core::status::SubscriptionMatchedStatus** &status)=0

  User callback.

8.280.1 Detailed Description

template<**typename T**>
class rti::queuing::QueueConsumerListener< **T** >

Called when certain events occur in a **QueueConsumer** (p. 2063).

A **QueueConsumer** (p. 2063) listener is a way to implement a callback that will be invoked when certain events happen. It is an optional parameter in **QueueConsumerParams** (p. 2077).

You can use this listener to receive notification when:

- Samples are available
- There are SharedReaderQueue matching events

See also

QueueConsumer::QueueConsumer() (p. 2066)

8.280.2 Member Function Documentation

8.280.2.1 on_sample_available()

```
template<typename T >
virtual void  rti::queuing::QueueConsumerListener< T >::on_sample_available (
              QueueConsumer< T > & consumer )  [pure virtual]
```

User callback.

This callback is invoked whenever the **QueueConsumer** (p. 2063) has received at least one sample. Any operation to get samples i.e. QueueConsumer::take_sample) can be called within this context.

See also

dds::sub::DataReaderListener::on_data_available (p. 1058)

Implemented in **rti::queuing::NoOpQueueConsumerListener< T >** (p. 1857), **rti::queuing::NoOpQueue↩ ConsumerListener< TRep >** (p. 1857), and **rti::queuing::NoOpQueueConsumerListener< TReq >** (p. 1857).

8.280.2.2 on_shared_reader_queue_matched()

```
template<typename T >
virtual void  rti::queuing::QueueConsumerListener< T >::on_shared_reader_queue_matched (
              QueueConsumer< T > & consumer,
             const  dds::core::status::SubscriptionMatchedStatus & status )  [pure virtual]
```

User callback.

This callback is invoked whenever a new SharedReaderQueue hosted by Queuing Service has matched the **Queue↩ Consumer** (p. 2063), or if an existing matching SharedReaderQueue is disposed.

See also

dds::core::status::SubscriptionMatchedStatus (p. 2439)

dds::sub::DataReaderListener::on_subscription_matched (p. 1058)

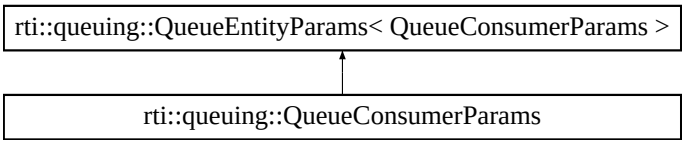
Implemented in **rti::queuing::NoOpQueueConsumerListener< T >** (p. 1857), **rti::queuing::NoOpQueue↩ ConsumerListener< TRep >** (p. 1857), and **rti::queuing::NoOpQueueConsumerListener< TReq >** (p. 1857).

8.281 rti::queuing::QueueConsumerParams Class Reference

Contains the parameters for creating a **QueueConsumer** (p. 2063).

`#include <rti/queuing/QueueParams.hpp>`

Inheritance diagram for rti::queuing::QueueConsumerParams:



Public Member Functions

- **QueueConsumerParams** (**dds::domain::DomainParticipant** participant)

  *Creates a **QueueConsumerParams** (p. 2077) object with the participant set.*
- **QueueConsumerParams** & **enable_availability** (bool enable)

  *Indicates whether the availability channel is enabled for a **QueueConsumer** (p. 2063).*

8.281.1 Detailed Description

Contains the parameters for creating a **QueueConsumer** (p. 2063).

8.281.2 Constructor & Destructor Documentation

8.281.2.1 QueueConsumerParams()

```
rti::queuing::QueueConsumerParams::QueueConsumerParams (
              dds::domain::DomainParticipant participant )  [inline], [explicit]
```

Creates a **QueueConsumerParams** (p. 2077) object with the participant set.

The rest of the parameters that can be set in a **QueueConsumerParams** (p. 2077) object are optional.

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) a **QueueConsumer** (p. 2063) uses to join a domain.

8.281.3 Member Function Documentation

8.281.3.1 enable_availability()

```
QueueConsumerParams & rti::queuing::QueueConsumerParams::enable_availability (
            bool enable )  [inline]
```

Indicates whether the availability channel is enabled for a **QueueConsumer** (p. 2063).

When this parameter is set to true, the **QueueConsumer** (p. 2063) creates a DataWriter to report its availability to the SharedReaderQueue from which the **QueueConsumer** (p. 2063) receives samples.

The **QueueConsumer** (p. 2063) can send availability samples to the SharedReaderQueue using the method **Queue↩ Consumer::send_availability(ConsumerAvailabilityParams)** (p. 2074)

See also

QueueConsumer::send_availability(ConsumerAvailabilityParams) (p. 2074)

References **rti::util::network_capture::enable()**.

8.282 rti::queuing::QueueEntityParams< ActualEntity > Class Template Reference

A parent class for all queue parameter classes.

`#include <rti/queuing/QueueParams.hpp>`

Inherits rti::request::detail::EntityParamsWithSetters< ActualEntity >.

Public Member Functions

- ActualEntity & **qos_profile** (const std::string &qos_library_name, const std::string &qos_profile_name)

  Specifies an XML QoS profile that will be used to configure the quality of service of the DDS entities created. This includes the DataReader and DataWriter used to send and receive samples from a SharedReaderQueue. Alternatively, you can set the DataReader and DataWriter QoS using datareader_qos() and datawriter_qos() methods. If you set the DataReader QoS or the DataWriter QoS the provided values will be used instead of the QoS specified in the XML qos profile. .
- ActualEntity & **shared_subscriber_name** (const std::string &name)

  Sets the SharedSubscriber name associated with the SharedReaderQueues.
- ActualEntity & **entity_name** (const std::string &name)

  Sets the name of the QueueEntity.

8.282.1 Detailed Description

template<typename ActualEntity>
class rti::queuing::QueueEntityParams< ActualEntity >

A parent class for all queue parameter classes.

8.282.2 Member Function Documentation

8.282.2.1 qos_profile()

```
template<typename ActualEntity >
ActualEntity &  rti::queuing::QueueEntityParams< ActualEntity >::qos_profile (
            const std::string & qos_library_name,
            const std::string & qos_profile_name )  [inline]
```

Specifies an XML QoS profile that will be used to configure the quality of service of the DDS entities created. This includes the DataReader and DataWriter used to send and receive samples from a SharedReaderQueue. Alternatively, you can set the DataReader and DataWriter QoS using datareader_qos() and datawriter_qos() methods. If you set the DataReader QoS or the DataWriter QoS the provided values will be used instead of the QoS specified in the XML qos profile. .

Parameters

qos_library_name	The name of the QoS library
qos_profile_name	The name of the QoS profile inside the QoS library

8.282.2.2 shared_subscriber_name()

```
template<typename ActualEntity >
ActualEntity &  rti::queuing::QueueEntityParams< ActualEntity >::shared_subscriber_name (
             const std::string & name )  [inline]
```

Sets the SharedSubscriber name associated with the SharedReaderQueues.

8.282.2.3 entity_name()

```
template<typename ActualEntity >
ActualEntity &  rti::queuing::QueueEntityParams< ActualEntity >::entity_name (
             const std::string & name )  [inline]
```

Sets the name of the QueueEntity.

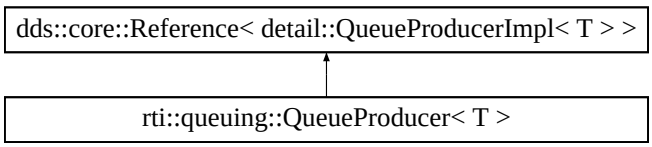
The name of the QueueEntity is used along with the topic name to generate the QueueEntity's GUID. The entity name is used along with the topic name to generate a Guid. By default, entity name is null which means that the entity Guid is randomly generated.

8.283 rti::queuing::QueueProducer< T > Class Template Reference

Allows you to send samples to a specific SharedReaderQueue hosted by Queuing Service.

```
#include <rti/queuing/QueueProducer.hpp>
```

Inheritance diagram for rti::queuing::QueueProducer< T >:



Public Member Functions

- **QueueProducer** (const **QueueProducerParams** &params, bool is_enabled=true, **Listener** *producer_listener=NULL, const **rti::core::Guid** &producer_guid= **rti::core::Guid::unknown**())

  Creates a queue producer.
- **QueueProducer** (const **QueueProducerParams** &params, bool is_enabled, std::shared_ptr< **Listener** > producer_listener, const **rti::core::Guid** &producer_guid= **rti::core::Guid::unknown**())

  *Creates a **QueueProducer** (p. 2080) with parameters.*
- void **listener** (**Listener** *the_listener, const **dds::core::status::StatusMask** &event_mask= **dds::core::status::StatusMask::none**())

  Sets the listener associated with this producer.
- **Listener** * **listener** () const

  Returns the listener currently associated with this Producer.
- std::shared_ptr< **Listener** > **get_listener** () const

  Gets the listener.
- void **set_listener** (std::shared_ptr< **Listener** > the_listener)

  Sets a listener to be notified of status updates.
- void **enable** ()

  *Enables the **QueueProducer** (p. 2080) to send data and receive listener notifications.*
- void **send_sample** (const T &sample)

  Sends a sample.
- void **send_sample** (const T &sample, **rti::pub::WriteParams** &write_params)

  Sends a sample with the specified parameters and gets back metadata information related to the sample sent.
- bool **wait_for_acknowledgments** (const **dds::core::Duration** &max_wait)

  *Blocks the calling thread until all samples written by this **QueueProducer** (p. 2080) since the last call to this method are acknowledged by Queuing Service.*
- bool **wait_for_acknowledgments** (const **rti::core::SampleIdentity** &identity, const **dds::core::Duration** &max_wait)

  Blocks the calling thread until the sample identified by identity is acknowledged by Queuing Service.
- **dds::pub::DataWriter**< T > **writer** () const

  *Retrieves the underlying **dds::pub::DataWriter** (p. 1138).*
- **rti::core::Guid** **guid** () const

  *Returns the GUID of this **QueueProducer** (p. 2080).*
- bool **has_matching_reader_queue** ()

  *Checks whether this **QueueProducer** (p. 2080) has matched with at least one SharedReaderQueue.*

8.283.1 Detailed Description

template<typename T>
class rti::queuing::QueueProducer< T >

Allows you to send samples to a specific SharedReaderQueue hosted by Queuing Service.

A **QueueProducer** (p. 2080) is an entity that allows you to send samples to a SharedReaderQueue hosted by Queuing Service. A **QueueProducer** (p. 2080) has one underlying **dds::pub::DataWriter** (p. 1138) to communicate with a SharedReaderQueue.

Valid types for the topic of the DataWriter (T) are those generated by rtiddsgen, the **DDS built-in types** (p. 50), and **dds::core::xtypes::DynamicData** (p. 1461).

To send samples to a SharedReaderQueue, a **QueueProducer** (p. 2080) must set its topic name (see QueueProducer↩ Params::queue_topic_name) equal to the SharedReaderQueue topic name.

The **QueueProducer** (p. 2080) and the SharedReadeeQueue must also be in the same DDS domain (that is, they must have the same domain ID).

A **QueueProducer** (p. 2080) has an associated **dds::domain::DomainParticipant** (p. 1326), which can be shared with other QueueProducers or RTI Connext routines. All the other RTI Connext entities required for queuing interaction, including a **dds::pub::DataWriter** (p. 1138) for writing samples, are automatically created when the **QueueProducer** (p. 2080) is constructed.

Quality of Service (QoS) for the underlying DataWriter is configurable (see **QueueProducerParams::qos_profile** (p. 2079) and QueueProducerParams::datawriter_qos).

If no QoS is specified in the **QueueProducerParams** (p. 2092), the default DataWriterQos will be used.

A **QueueProducer** (p. 2080) must always be reliable in order to match and communicate with the SharedReaderQueue. Hence, whichever DataWriterQos is selected, the following QoS setting is always overriden:

- **dds::core::policy::Reliability::kind** (p. 2153) is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157).

The underlying DataWriter is created with an internal **dds::pub::DataWriterListener** (p. 1202), which is required to provide the behavior of a **QueueProducer** (p. 2080), including the notification of events via **QueueProducerListener** (p. 2090). For this reason, when accessing the underlying DataWriter, the listener should not be set, removed or modified; otherwise the behavior of the **QueueProducer** (p. 2080) will be incorrect and unpredictable.

Template Parameters

T	The data type for the SharedReaderQueue topic

See also

QueueProducer (p. 2080)

QueueProducerListener (p. 2090)

8.283.2 Constructor & Destructor Documentation

8.283.2.1 QueueProducer() [1/2]

```
template<typename T >
rti::queuing::QueueProducer< T >::QueueProducer (
```

```
            const  QueueProducerParams & params,
            bool is_enabled = true,
             Listener * producer_listener = NULL,
            const  rti::core::Guid & producer_guid =  rti::core::Guid::unknown() )  [inline],
[explicit]
```

Creates a queue producer.

[DEPRECATED] When using a listener, prefer the constructor that receives a shared_ptr<**Listener** (p. 1637)> instead of a regular Listener* pointer.

8.283.2.2 QueueProducer() [2/2]

```
template<typename T >
rti::queuing::QueueProducer< T >::QueueProducer (
            const  QueueProducerParams & params,
            bool is_enabled,
            std::shared_ptr<  Listener > producer_listener,
            const  rti::core::Guid & producer_guid =  rti::core::Guid::unknown() )  [inline],
[explicit]
```

Creates a **QueueProducer** (p. 2080) with parameters.

Parameters

params	All the parameters that configure this **QueueProducer** (p. 2080). See **QueueProducerParams** (p. 2092) for the list of required parameters.
is_enabled	Specifies if the **QueueProducer** (p. 2080) is created ready to receive notifications in its listener and send data. If you choose to bind the **QueueProducer** (p. 2080) to a **QueueProducerListener** (p. 2090) using an **rti::core::ListenerBinder** (p. 1641) you should create the **QueueProducer** (p. 2080) disabled to avoid missing listener notifications. You can later enable the disabled **QueueProducer** (p. 2080) using **QueueProducer::enable** (p. 2085).
producer_listener	A shared_ptr to a **QueueProducerListener** (p. 2090) object to receive event notifications.
producer_guid	A GUID identifier for the producer. When producer_guid is set to **rti::core::Guid::unknown()** (p. 1597) the Guid identifier is generated automatically. You can retrieve the generated **QueueProducer** (p. 2080) Guid using the **QueueProducer::guid** (p. 2089) method. To restart a **QueueProducer** (p. 2080) just create a new one with its same GUID.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

QueueProducerParams (p. 2092)

8.283.3 Member Function Documentation

8.283.3.1 listener() [1/2]

```
template<typename T >
void  rti::queuing::QueueProducer< T >::listener (
             Listener * the_listener,
            const  dds::core::status::StatusMask & event_mask =  dds::core::status::StatusMask↩
::none() )  [inline]
```

Sets the listener associated with this producer.

[DEPRECATED] The use of **set_listener()** (p. 2084) is recommended. Unlike this function, set_listener receives a shared_ptr which simplifies the management of listener's lifecycle.

Parameters

the_listener	The **QueueProducerListener** (p. 2090) to set
event_mask	The **dds::core::status::StatusMask** (p. 2375) associated with the listener

8.283.3.2 listener() [2/2]

```
template<typename T >
Listener *  rti::queuing::QueueProducer< T >::listener ( ) const  [inline]
```

Returns the listener currently associated with this Producer.

[DEPRECATED] Prefer **get_listener()** (p. 2084) instead of this function.

If there is no listener it returns NULL.

8.283.3.3 get_listener()

```
template<typename T >
std::shared_ptr<  Listener >  rti::queuing::QueueProducer< T >::get_listener ( ) const  [inline]
```

Gets the listener.

8.283.3.4 set_listener()

```
template<typename T >
void  rti::queuing::QueueProducer< T >::set_listener (
            std::shared_ptr<  Listener > the_listener )  [inline]
```

Sets a listener to be notified of status updates.

Warning

It's recommended that the listener implementation doesn't hold a permanent reference to this object. If it does, the application needs to manually reset the listener or manually close this object to ensure that there is no cycle that prevents the destruction of these two objects.

Parameters

the_listener	A shared pointer to the listener to receive updates or `nullptr` to reset the current listener and stop receiving updates.

8.283.3.5 enable()

```
template<typename T >
void  rti::queuing::QueueProducer< T >::enable ( )  [inline]
```

Enables the **QueueProducer** (p. 2080) to send data and receive listener notifications.

If you create the **QueueProducer** (p. 2080) disabled you can enable it using this mehtod. To avoid missing listener notifciation when you use an **rti::core::ListenerBinder** (p. 1641) to bind the **QueueProducer** (p. 2080) to a **Queue↩ ProducerListener** (p. 2090) use **QueueProducer::enable** (p. 2085) to enable the **QueueProducer** (p. 2080) after it is bound to the listener.

8.283.3.6 send_sample() [1/2]

```
template<typename T >
void  rti::queuing::QueueProducer< T >::send_sample (
            const T & sample )  [inline]
```

Sends a sample.

This operation is equivalent to calling **QueueProducer::send_sample(const T&, rti::pub::WriteParams&)** (p. 2085) using a default **rti::pub::WriteParams** (p. 2642).

Parameters

sample	The sample to be sent

Exceptions

One	of the **Standard Exceptions** (p. 428) or **NoMatchingQueueException** (p. 1835)

MT Safety:

SAFE

See also

QueueProducer::send_sample(const T&, rti::pub::WriteParams&) (p. 2085)

8.283.3.7 send_sample() [2/2]

```
template<typename T >
void  rti::queuing::QueueProducer< T >::send_sample (
            const T & sample,
             rti::pub::WriteParams & write_params )  [inline]
```

Sends a sample with the specified parameters and gets back metadata information related to the sample sent.

After calling this operation, write_params contains valid metadata information associated with the sample sent.

The metadata field **rti::pub::WriteParams::source_guid** (p. 2650) of the write_params in the sample is always set to this **QueueProducer** (p. 2080)'s GUID.

Parameters

sample	<<***in***>> *(p. 355)* the sample to send.
write_params	<<***inout***>> *(p. 355)* Parameters to send the sample. When this call ends successfully, `write_params` contains valid metadata information associated with the sample sent.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **NoMatchingQueueException** (p. 1835) if by the time this operation is called there is not a single matching SharedReaderQueue.

MT Safety:

SAFE

See also

QueueProducer::send_sample(const T&) (p. 2085)

8.283.3.8 wait_for_acknowledgments() [1/2]

```
template<typename T >
bool  rti::queuing::QueueProducer< T >::wait_for_acknowledgments (
            const  dds::core::Duration & max_wait )  [inline]
```

Blocks the calling thread until all samples written by this **QueueProducer** (p. 2080) since the last call to this method are acknowledged by Queuing Service.

This operation will keep the calling thread blocked as long as there are samples pending on acknowledgement, unless a timeout occurs.

After unblocking, the return value is true if all samples have been acknowledged positively or false if at least one of them was acknowledged negatively.

If there is a timeout, this operation throws **dds::core::TimeoutError** (p. 2473).

This operation requires setting QueueProducerParams::enable_wait_for_ack to true. Otherwise, it will throw **dds::core::PreconditionNotMetError** (p. 1941).

When a SharedReaderQueue is not replicated, a sample is acknowledged positively once the running Queuing Service instance has enqueued the sample.

When a SharedReaderQueue is replicated, a sample is acknowledged positively once it has been enqueued by the Queuing Service instance acting as a master. This means that the sample has been received and enqueued by a configurable quorum of instances (using <replication_settings>/<queue_instances>).

This operation waits until all the previously written samples since the last call to this method are acknowledged. If you want to wait for a specific sample, use the method **QueueProducer::wait_for_acknowledgments(const rti::core::SampleIdentity&, const dds::core::Duration&)** (p. 2087).

This operation cannot be called by two threads concurrently. If this occurs, the second thread will throw **dds::core::PreconditionNotMetError** (p. 1941).

This operation cannot be called concurrently with **QueueProducer::wait_for_acknowledgments(const rti::core::SampleIdentity&, const dds::core::Duration&)** (p. 2087) either.

Parameters

max_wait	<<***in***>> *(p. 355)* Maximum waiting time.

Returns

true if all the samples are acknowledged positively or false if there was at least one sample acknowledged negatively.

Exceptions

One	of the **Standard Exceptions** (p. 428) ; **NoMatchingQueueException** (p. 1835) if during this call, the **QueueProducer** (p. 2080) lost the matching status with Queuing Service. **dds::core::TimeoutError** (p. 2473) if a timeout occurs before receiving all the acknowledgments.

8.283.3.9 wait_for_acknowledgments() [2/2]

```
template<typename T >
bool  rti::queuing::QueueProducer< T >::wait_for_acknowledgments (
            const  rti::core::SampleIdentity & identity,
            const  dds::core::Duration & max_wait )  [inline]
```

Blocks the calling thread until the sample identified by identity is acknowledged by Queuing Service.

This operation blocks the calling thread until the sample identified by identity is acknowledged by Queuing Service or there is a timeout.

After unblocking, the return value is true if the sample has been acknowledged positively or false if the sample has been acknowledged negatively.

If there is a timeout, this operation throws **dds::core::TimeoutError** (p. 2473).

This operation requires setting QueueProducerParams::enable_wait_for_ack to true. Otherwise, it will throw **dds::core::PreconditionNotMetError** (p. 1941).

When a SharedReaderQueue is not replicated, a sample is acknowledged positively once the running Queuing Service instance has enqueued the sample.

When a SharedReaderQueue is replicated, a sample is acknowledged positively once it has been enqueued by the Queuing Service instance acting as a master. This means that the sample has been received and enqueued by a configurable quorum of instances (using <replication_settings>/<queue_instances>).

This operation cannot be called by two threads concurrently on the same identity. If this occurs, the second thread will throw **dds::core::PreconditionNotMetError** (p. 1941).

This operation cannot be called concurrently with **QueueProducer::wait_for_acknowledgments(const dds::core::Duration&)** (p. 2086) either.

Parameters

max_wait	<<***in***>> *(p. 355)* Maximum waiting time.
identity	<<***in***>> *(p. 355)*. Sample identity.

Returns

true if the sample is acknowledged positively or false if the sample is acknowledged negatively.

Exceptions

One	of the **Standard Exceptions** (p. 428) ; **NoMatchingQueueException** (p. 1835) if during this call, the **QueueProducer** (p. 2080) lost the matching status with Queuing Service. **dds::core::TimeoutError** (p. 2473) if a timeout occurs before receiving all the acknowledgments.

8.283.3.10 writer()

```
template<typename T >
dds::pub::DataWriter< T >  rti::queuing::QueueProducer< T >::writer ( ) const  [inline]
```

Retrieves the underlying **dds::pub::DataWriter** (p. 1138).

Accessing the DataWriter may be useful for a number of advanced use cases, such as getting the DataWriter protocol or cache statuses.

MT Safety:

SAFE

See also

dds::pub::DataWriter (p. 1138)

dds::pub::DataWriter (p. 1138)

dds::pub::DataWriter::datawriter_protocol_status() (p. 1176)

8.283.3.11 guid()

```
template<typename T >
rti::core::Guid  rti::queuing::QueueProducer< T >::guid ( ) const  [inline]
```

Returns the GUID of this **QueueProducer** (p. 2080).

The GUID of the **QueueProducer** (p. 2080) is determined based on the value of QueueEntityParams::entity_name and QueueProducerParams::queue_topic_name. Note that the **QueueProducer** (p. 2080) GUID may be equal to or different than the GUID of the underlying DataWriter virtual GUID.

The GUID identifies a **QueueProducer** (p. 2080) and the samples it writes. This is the value set in the source GUID of the samples sent by the **QueueProducer** (p. 2080).

See also

QueueProducer::send_sample(const T&) (p. 2085)

QueueEntityParams::entity_name

8.283.3.12 has_matching_reader_queue()

```
template<typename T >
bool  rti::queuing::QueueProducer< T >::has_matching_reader_queue ( )  [inline]
```

Checks whether this **QueueProducer** (p. 2080) has matched with at least one SharedReaderQueue.

Exceptions

One	of the **Standard Exceptions** (p. 428) ;

Returns

True if this **QueueProducer** (p. 2080) matches with at least one SharedReaderQueue.

MT Safety:

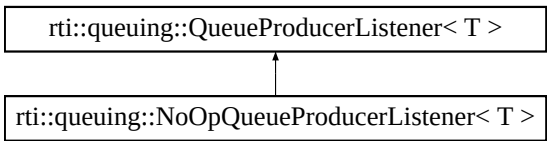
SAFE

8.284 rti::queuing::QueueProducerListener< T > Class Template Reference

Called when certain events occur in a **QueueProducer** (p. 2080).

`#include <rti/queuing/QueueProducerListener.hpp>`

Inheritance diagram for rti::queuing::QueueProducerListener< T >:



Public Member Functions

- virtual void **on_sample_acknowledged** (**QueueProducer**< T > &producer, const **rti::pub::AcknowledgmentInfo** &info)=0

  User callback.
- virtual void **on_shared_reader_queue_matched** (**QueueProducer**< T > &producer, const **dds::core::status::PublicationMatchedStatus** &status)=0

  User callback.

8.284.1 Detailed Description

template<typename T>
class rti::queuing::QueueProducerListener< T >

Called when certain events occur in a **QueueProducer** (p. 2080).

A **QueueProducer** (p. 2080) listener is a way to implement a callback that will be invoked when certain events happen. It is an optional parameter in **QueueProducerParams** (p. 2092).

You can use this listener to receive notification when:

- A sample previously sent by a **QueueProducer** (p. 2080) has been acknowledged by Queuing Service.
- There are SharedReaderQueue matching events.

See also

QueueProducer::QueueProducer() (p. 2082)

8.284.2 Member Function Documentation

8.284.2.1 on_sample_acknowledged()

```
template<typename T >
virtual void  rti::queuing::QueueProducerListener< T >::on_sample_acknowledged (
              QueueProducer< T > & producer,
             const  rti::pub::AcknowledgmentInfo & info )  [pure virtual]
```

User callback.

This callback is invoked whenever Queuing Service acknowledges a sample sent by the **QueueProducer** (p. 2080).

See also

AcknowledgmentInfo

Implemented in **rti::queuing::NoOpQueueProducerListener< T >** (p. 1858), **rti::queuing::NoOpQueue↩ ProducerListener< TRep >** (p. 1858), and **rti::queuing::NoOpQueueProducerListener< TReq >** (p. 1858).

8.284.2.2 on_shared_reader_queue_matched()

```
template<typename T >
virtual void  rti::queuing::QueueProducerListener< T >::on_shared_reader_queue_matched (
              QueueProducer< T > & producer,
             const  dds::core::status::PublicationMatchedStatus & status )  [pure virtual]
```

User callback.

This callback is invoked whenever a new SharedReaderQueue hosted by Queuing Service has matched the **QueueProducer** (p. 2080), or if an existing matching SharedReaderQueue is disposed.

See also

dds::core::status::PublicationMatchedStatus (p. 1990)

dds::pub::DataWriterListener::on_publication_matched (p. 1205)

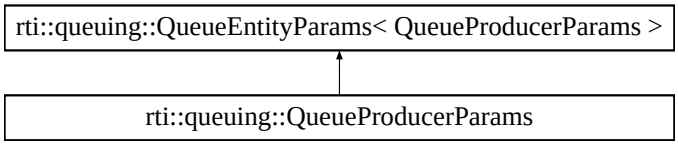
Implemented in **rti::queuing::NoOpQueueProducerListener< T >** (p. 1858), **rti::queuing::NoOpQueueProducerListener< TRep >** (p. 1858), and **rti::queuing::NoOpQueueProducerListener< TReq >** (p. 1858).

8.285 rti::queuing::QueueProducerParams Class Reference

Contains the parameters for creating a **QueueProducer** (p. 2080).

`#include <rti/queuing/QueueParams.hpp>`

Inheritance diagram for rti::queuing::QueueProducerParams:



Public Member Functions

- **QueueProducerParams** (**dds::domain::DomainParticipant** participant)

  *Creates a **QueueProducerParams** (p. 2092) object with the participant set.*
- bool **enable_sample_replication** () const

  *Causes the **QueueProducer** (p. 2080) to write all the samples with the **dds::sub::SampleInfo::flag** (p. 2293) including the bit **rti::core::SampleFlag::replicate** (p. 2280).*
- **QueueProducerParams** & **enable_wait_for_ack** (bool enable)

  *Enables the Queuing Service's 'Acknowledgment Management' feature for the **QueueProducer** (p. 2080).*

8.285.1 Detailed Description

Contains the parameters for creating a **QueueProducer** (p. 2080).

8.285.2 Constructor & Destructor Documentation

8.285.2.1 QueueProducerParams()

```
rti::queuing::QueueProducerParams::QueueProducerParams (
            dds::domain::DomainParticipant participant )  [inline], [explicit]
```

Creates a **QueueProducerParams** (p. 2092) object with the participant set.

The rest of the parameters that can be set in a **QueueProducerParams** (p. 2092) object are optional.

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) a **QueueProducer** (p. 2080) uses to join a domain.

8.285.3 Member Function Documentation

8.285.3.1 enable_sample_replication()

```
bool rti::queuing::QueueProducerParams::enable_sample_replication ( ) const  [inline]
```

Causes the **QueueProducer** (p. 2080) to write all the samples with the **dds::sub::SampleInfo::flag** (p. 2293) including the bit **rti::core::SampleFlag::replicate** (p. 2280).

When this value is set to true, all the samples are written with the associated metadata containing the bit **rti::core::SampleFlag::replicate** (p. 2280). This behavior cannot be changed once this option is enabled (no matter which operation is used to send samples).

8.285.3.2 enable_wait_for_ack()

```
QueueProducerParams & rti::queuing::QueueProducerParams::enable_wait_for_ack (
            bool enable )  [inline]
```

Enables the Queuing Service's 'Acknowledgment Management' feature for the **QueueProducer** (p. 2080).

When this value is set to true, you can call the APIs **QueueProducer::wait_for_acknowledgments(const dds::core::Duration&)** (p. 2086) and **QueueProducer::wait_for_acknowledgments(const rti::core::SampleIdentity&, const dds::core::Duration&)** (p. 2087) to wait for acknowledgments from Queuing Service. These acknowledgments indicate whether or not samples have been successfully enqueued.

Notice that if you enable this feature, at some point you must call **QueueProducer::wait_for_acknowledgments(const dds::core::Duration&)** (p. 2086) or **QueueProducer::wait_for_acknowledgments(const rti::core::SampleIdentity&, const dds::core::Duration&)** (p. 2087). Otherwise, the **QueueProducer** (p. 2080)'s memory will grow unbounded, since it will keep some state per sample.

You must also set the XML tag <app_ack_sample_to_producer> under <queue_qos>/<reliability> to true.

[default] true

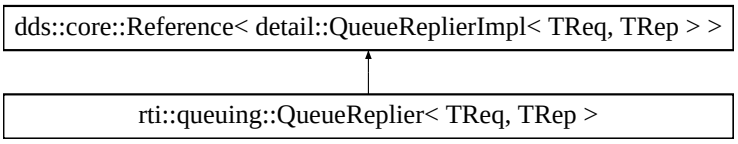
References **rti::util::network_capture::enable()**.

8.286 rti::queuing::QueueReplier< TReq, TRep > Class Template Reference

Allows receiving requests and sending replies.

`#include <rti/queuing/QueueReplier.hpp>`

Inheritance diagram for rti::queuing::QueueReplier< TReq, TRep >:



Public Member Functions

- **QueueReplier** (const **QueueReplierParams** &params, bool is_enabled=true, **Listener** *replier_listener=NULL, const **rti::core::Guid** &replier_guid= **rti::core::Guid::unknown**())

  Creates a queue replier.
- **QueueReplier** (const **QueueReplierParams** &params, bool is_enabled, std::shared_ptr< **Listener** > replier_listener, const **rti::core::Guid** &replier_guid= **rti::core::Guid::unknown**())

  Creates a Replier with parameters.
- void **listener** (**Listener** *the_listener, const **dds::core::status::StatusMask** &event_mask= **dds::core::status::StatusMask::none**())

  Sets the listener associated with this queue requester.
- **Listener** * **listener** () const

  Returns the listener currently associated with this requester.
- std::shared_ptr< **Listener** > **get_listener** () const

  Gets the listener.
- void **set_listener** (std::shared_ptr< **Listener** > the_listener)

  Sets a listener to be notified of status updates.
- void **enable** ()

  *Enables the **QueueReplier** (p. 2094) to send data, receive data and receive listener notifications.*
- void **send_reply** (const TRep &reply, const **dds::sub::SampleInfo** &related_request_info)

  Sends a reply for a previous request.
- void **send_reply** (const TRep &reply, **rti::pub::WriteParams** &write_params)

  Sends a reply with the specified parameters and gets back metadata information related to the reply sent.
- void **acknowledge_request** (const **dds::sub::SampleInfo** &sample_info, bool is_positive_acknowledgment=true)

  Explicitly acknowledges a single request.
- void **acknowledge_all** (bool is_positive_acknowledgment=true)

  Acknowledges all previously accessed requests.
- bool **wait_for_acknowledgments** (const **dds::core::Duration** &max_wait)

  *Blocks the calling thread until all replies written by this **QueueReplier** (p. 2094) are acknowledged by Queuing Service, or until a timeout occurs.*
- bool **wait_for_acknowledgments** (const **rti::core::SampleIdentity** &identity, const **dds::core::Duration** &max_wait)

Blocks the calling thread until the reply identified by identity is acknowledged by Queuing Service, or until a timeout occurs.

- **rti::queuing::QueueConsumer**< TReq > **consumer** () const

  *Retrieves the underlying **dds::sub::DataReader** (p. 976).*
- **rti::queuing::QueueProducer**< TRep > **producer** () const

  *Retrieves the underlying **dds::pub::DataWriter** (p. 1138).*
- void **send_availability** (**ConsumerAvailabilityParams** parameters)

  Sends an availability sample with the specified parameters. The availability applies to the reply queue.
- bool **has_matching_request_reader_queue** ()

  *Checks whether or not this **QueueReplier** (p. 2094) has matched with at least one request SharedReaderQueue.*
- bool **has_matching_reply_reader_queue** ()

  *Checks whether or not this **QueueReplier** (p. 2094) has matched with at least one reply SharedReaderQueue.*
- bool **wait_for_requests** (const **dds::core::Duration** &max_wait)

  Waits for requests.
- bool **wait_for_requests** (int min_count, const **dds::core::Duration** &max_wait)

  Waits for requests.
- **dds::sub::LoanedSamples**< TReq > **receive_requests** (const **dds::core::Duration** &max_wait)

  Waits for multiple requests and provides a loaned container to access them.
- **dds::sub::LoanedSamples**< TReq > **receive_requests** (int min_count, int max_count, const **dds::core::Duration** &max_wait)

  Waits for multiple requests and provides a loaned container to access them.
- **rti::pub::WriteParams get_write_params_for_related_request** (**rti::pub::WriteParams** &write_params, const **dds::sub::SampleInfo** &related_request_info)

  Gets the write parameters ready to send a reply for a given related request.
- **dds::sub::LoanedSamples**< TReq > **take_requests** ()

  Provides a loaned container to access the existing requests.
- **dds::sub::LoanedSamples**< TReq > **take_requests** (int max_count)

  Provides a loaned container to access the existing requests.
- **dds::sub::LoanedSamples**< TReq > **read_requests** ()

  Provides a loaned container from which you can access the existing requests.
- **dds::sub::LoanedSamples**< TReq > **read_requests** (int max_count)

  Provides a loaned container from which you can access the existing requests.
- **dds::pub::DataWriter**< TRep > **writer** () const

  *Retrieves the underlying **dds::pub::DataWriter** (p. 1138) used to send replies.*
- **dds::sub::DataReader**< TReq > **reader** () const

  *Retrieves the underlying **dds::sub::DataReader** (p. 976) used to receive requests.*
- **rti::core::Guid guid** ()

  *Returns the GUID of this **QueueReplier** (p. 2094).*

8.286.1 Detailed Description

template<typename TReq, typename TRep>
class rti::queuing::QueueReplier< TReq, TRep >

Allows receiving requests and sending replies.

A **QueueReplier** (p. 2094) is a component suited for the queuing request-reply use case. A **QueueReplier** (p. 2094) plays the role of a request **QueueConsumer** (p. 2063) and reply **QueueProducer** (p. 2080).

A **QueueReplier** (p. 2094) is an entity that allows you to receive requests from a request SharedReaderQueue and send replies to a reply SharedReaderQueue, which are hosted by Queuing Service.

To communicate with the SharedReaderQueues, the **QueueReplier** (p. 2094) creates two topics. The first topic corresponds to the request SharedReaderQueue topic and the second topic to the reply SharedReaderQueue topic.

Valid types for these topics (`TReq` and `TRep`) are: those generated by rtiddsgen, the **DDS built-in types** (p. 50), and **dds::core::xtypes::DynamicData** (p. 1461).

See also

QueueConsumer (p. 2063)

QueueProducer (p. 2080)

Template Parameters

TReq	The data type for the request SharedReaderQueue topic
TRep	The data type for the reply SharedReaderQueue topic

8.286.2 Constructor & Destructor Documentation

8.286.2.1 QueueReplier() [1/2]

```
template<typename TReq , typename TRep >
rti::queuing::QueueReplier< TReq, TRep >::QueueReplier (
            const  QueueReplierParams & params,
            bool is_enabled = true,
             Listener * replier_listener = NULL,
            const  rti::core::Guid & replier_guid =  rti::core::Guid::unknown() )  [inline],
[explicit]
```

Creates a queue replier.

[DEPRECATED] When using a listener, prefer the constructor that receives a `shared_ptr<`**Listener** (p. 1637)`>` instead of a regular `Listener*` pointer.

8.286.2.2 QueueReplier() [2/2]

```
template<typename TReq , typename TRep >
rti::queuing::QueueReplier< TReq, TRep >::QueueReplier (
            const  QueueReplierParams & params,
            bool is_enabled,
            std::shared_ptr<  Listener > replier_listener,
            const  rti::core::Guid & replier_guid =  rti::core::Guid::unknown() )  [inline],
[explicit]
```

Creates a Replier with parameters.

Parameters

params	All the parameters that configure this **QueueReplier** (p. 2094) See **QueueReplierParams** (p. 2109) for the list of required parameters.
is_enabled	Specifies if the **QueueReplier** (p. 2094) is created ready to receive notifications in its listener and send or receive data. If you choose to bind the **QueueReplier** (p. 2094) to a **QueueReplierListener** (p. 2107) using an **rti::core::ListenerBinder** (p. 1641) you should create the **QueueReplier** (p. 2094) disabled to avoid missing listener notifications. You can later enable the disabled **QueueReplier** (p. 2094) using **QueueReplier::enable** (p. 2098).
replier_listener	A shared_ptr to a **QueueReplierListener** (p. 2107) object to receive event notifications.
replier_guid	A GUID identifier for the replier. When replier_guid is set to **rti::core::Guid::unknown()** (p. 1597) the Guid identifier is generated automatically. You can retrieve the generated **QueueReplier** (p. 2094) Guid using the **QueueReplier::guid()** (p. 2106) method. To restart a **QueueReplier** (p. 2094) just create a new one with its same GUID.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

QueueReplierParams (p. 2109)

8.286.3 Member Function Documentation

8.286.3.1 listener() [1/2]

```
template<typename TReq , typename TRep >
void  rti::queuing::QueueReplier< TReq, TRep >::listener (
              Listener * the_listener,
             const  dds::core::status::StatusMask & event_mask =  dds::core::status::StatusMask↩
::none() )  [inline]
```

Sets the listener associated with this queue requester.

[DEPRECATED] The use of **`set_listener()`** (p. 2098) is recommended. Unlike this function, `set_listener` receives a `shared_ptr` which simplifies the management of listener's lifecycle.

Parameters

the_listener	The **QueueReplierListener** (p. 2107) to set
event_mask	The **dds::core::status::StatusMask** (p. 2375) associated with the listener

8.286.3.2 listener() [2/2]

```
template<typename TReq , typename TRep >
Listener * rti::queuing::QueueReplier< TReq, TRep >::listener ( ) const [inline]
```

Returns the listener currently associated with this requester.

[DEPRECATED] Prefer **get_listener()** (p. 2098) instead of this function.

If there is no listener it returns NULL.

8.286.3.3 get_listener()

```
template<typename TReq , typename TRep >
std::shared_ptr< Listener > rti::queuing::QueueReplier< TReq, TRep >::get_listener ( ) const
[inline]
```

Gets the listener.

8.286.3.4 set_listener()

```
template<typename TReq , typename TRep >
void rti::queuing::QueueReplier< TReq, TRep >::set_listener (
            std::shared_ptr< Listener > the_listener ) [inline]
```

Sets a listener to be notified of status updates.

Warning

It's recommended that the listener implementation doesn't hold a permanent reference to this object. If it does, the application needs to manually reset the listener or manually close this object to ensure that there is no cycle that prevents the destruction of these two objects.

Parameters

the_listener	A shared pointer to the listener to receive updates or `nullptr` to reset the current listener and stop receiving updates.

8.286.3.5 enable()

```
template<typename TReq , typename TRep >
void rti::queuing::QueueReplier< TReq, TRep >::enable ( ) [inline]
```

Enables the **QueueReplier** (p. 2094) to send data, receive data and receive listener notifications.

If you create the **QueueReplier** (p. 2094) disabled you can enable it using this mehtod. To avoid missing listener notifciation when you use an **rti::core::ListenerBinder** (p. 1641) to bind the **QueueReplier** (p. 2094) to a **QueueReplierListener** (p. 2107) use **QueueReplier::enable** (p. 2098) to enable the **QueueReplier** (p. 2094) after it is bound to the listener.

8.286.3.6 send_reply() [1/2]

```
template<typename TReq , typename TRep >
void  rti::queuing::QueueReplier< TReq, TRep >::send_reply (
            const TRep & reply,
            const  dds::sub::SampleInfo & related_request_info )  [inline]
```

Sends a reply for a previous request.

The related request identity can be retrieved from an existing request sample (Sample).

Parameters

reply	The reply to be sent
related_request_info	The identity of a previously received request

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

QueueProducer::send_sample(const T&) (p. 2085)

8.286.3.7 send_reply() [2/2]

```
template<typename TReq , typename TRep >
void  rti::queuing::QueueReplier< TReq, TRep >::send_reply (
            const TRep & reply,
             rti::pub::WriteParams & write_params )  [inline]
```

Sends a reply with the specified parameters and gets back metadata information related to the reply sent.

After calling this operation, write_params contains metadata information associated with the reply sent.

Contrary to the **QueueRequester** (p. 2111), where retrieving the sample identity for correlation is common, on the **QueueReplier** (p. 2094) side using a write_params is only necessary when the default write parameters need to be overridden, and **QueueReplier::send_reply(const TRep&, const dds::sub::SampleInfo&)** (p. 2099) may be used if that is not necessary.

Parameters

reply	<<***in***>> *(*p. *355)* the reply sample to send.
write_params	<<***inout***>> *(*p. *355)* Parameters used to send the reply. When this call ends successfully, `write_params` contains valid metadata information associated with the reply sent.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **NoMatchingQueueException** (p. 1835) if by the time this operation is called there is not a single matching SharedReaderQueue.

MT Safety:

SAFE

See also

QueueReplier::send_reply(const TRep&, const dds::sub::SampleInfo&) (p. 2099)

QueueProducer::send_sample(const T&, rti::pub::WriteParams&) (p. 2085)

QueueRequester::wait_for_replies(int, const dds::core::Duration&, const rti::core::SampleIdentity&) (p. 2123)

QueueReplier::take_requests(int) (p. 2105)

8.286.3.8 acknowledge_request()

```
template<typename TReq , typename TRep >
void  rti::queuing::QueueReplier< TReq, TRep >::acknowledge_request (
            const  dds::sub::SampleInfo & sample_info,
            bool is_positive_acknowledgment = true )  [inline]
```

Explicitly acknowledges a single request.

See also

QueueConsumer::acknowledge_sample(const dds::sub::SampleInfo&, bool) (p. 2069)

QueueReplier::acknowledge_all(bool) (p. 2100)

8.286.3.9 acknowledge_all()

```
template<typename TReq , typename TRep >
void  rti::queuing::QueueReplier< TReq, TRep >::acknowledge_all (
            bool is_positive_acknowledgment = true )  [inline]
```

Acknowledges all previously accessed requests.

See also

QueueConsumer::acknowledge_all(bool) (p. 2069)

QueueReplier::acknowledge_request(const dds::sub::SampleInfo&, bool) (p. 2100)

8.286.3.10 wait_for_acknowledgments() [1/2]

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueReplier< TReq, TRep >::wait_for_acknowledgments (
            const  dds::core::Duration & max_wait )  [inline]
```

Blocks the calling thread until all replies written by this **QueueReplier** (p. 2094) are acknowledged by Queuing Service, or until a timeout occurs.

See also

QueueProducer::wait_for_acknowledgments(const dds::core::Duration&) (p. 2086)

QueueReplier::send_reply(const TRep&, const dds::sub::SampleInfo&) (p. 2099)

QueueReplierListener::on_reply_acknowledged (p. 2108)

8.286.3.11 wait_for_acknowledgments() [2/2]

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueReplier< TReq, TRep >::wait_for_acknowledgments (
            const  rti::core::SampleIdentity & identity,
            const  dds::core::Duration & max_wait )  [inline]
```

Blocks the calling thread until the reply identified by identity is acknowledged by Queuing Service, or until a timeout occurs.

See also

QueueProducer::wait_for_acknowledgments(const rti::core::SampleIdentity&, const dds::core::↩ Duration&) (p. 2087)

QueueReplier::send_reply(const TRep&, const dds::sub::SampleInfo&) (p. 2099)

QueueReplierListener::on_reply_acknowledged (p. 2108)

8.286.3.12 consumer()

```
template<typename TReq , typename TRep >
rti::queuing::QueueConsumer< TReq >  rti::queuing::QueueReplier< TReq, TRep >::consumer ( )
const  [inline]
```

Retrieves the underlying **dds::sub::DataReader** (p. 976).

8.286.3.13 producer()

```
template<typename TReq , typename TRep >
rti::queuing::QueueProducer< TRep >  rti::queuing::QueueReplier< TReq, TRep >::producer ( )
const  [inline]
```

Retrieves the underlying **dds::pub::DataWriter** (p. 1138).

8.286.3.14 send_availability()

```
template<typename TReq , typename TRep >
void  rti::queuing::QueueReplier< TReq, TRep >::send_availability (
              ConsumerAvailabilityParams parameters )  [inline]
```

Sends an availability sample with the specified parameters. The availability applies to the reply queue.

See also

QueueConsumer::send_availability(ConsumerAvailabilityParams) (p. 2074)

QueueReplierParams::enable_availability

8.286.3.15 has_matching_request_reader_queue()

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueReplier< TReq, TRep >::has_matching_request_reader_queue ( )  [inline]
```

Checks whether or not this **QueueReplier** (p. 2094) has matched with at least one request SharedReaderQueue.

See also

QueueProducer::has_matching_reader_queue() (p. 2089)

8.286.3.16 has_matching_reply_reader_queue()

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueReplier< TReq, TRep >::has_matching_reply_reader_queue ( )  [inline]
```

Checks whether or not this **QueueReplier** (p. 2094) has matched with at least one reply SharedReaderQueue.

See also

QueueConsumer::has_matching_reader_queue() (p. 2075)

8.286.3.17 wait_for_requests() [1/2]

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueReplier< TReq, TRep >::wait_for_requests (
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for requests.

See also

QueueConsumer::wait_for_samples(int, const dds::core::Duration&) (p. 2073)

8.286.3.18 wait_for_requests() [2/2]

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueReplier< TReq, TRep >::wait_for_requests (
            int min_count,
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for requests.

See also

QueueConsumer::wait_for_samples(int, const dds::core::Duration&) (p. 2073)

8.286.3.19 receive_requests() [1/2]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TReq >  rti::queuing::QueueReplier< TReq, TRep >::receive_requests (
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for multiple requests and provides a loaned container to access them.

See also

QueueConsumer::receive_samples(const dds::core::Duration&) (p. 2071)

QueueReplier::wait_for_requests(int, const dds::core::Duration&) (p. 2103)

QueueReplier::take_requests(int) (p. 2105)

8.286.3.20 receive_requests() [2/2]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TReq >  rti::queuing::QueueReplier< TReq, TRep >::receive_requests (
            int min_count,
            int max_count,
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for multiple requests and provides a loaned container to access them.

See also

QueueConsumer::receive_samples(int, int, const dds::core::Duration&) (p. 2071)

QueueReplier::wait_for_requests(int, const dds::core::Duration&) (p. 2103)

QueueReplier::take_requests(int) (p. 2105)

8.286.3.21 get_write_params_for_related_request()

```
template<typename TReq , typename TRep >
rti::pub::WriteParams  rti::queuing::QueueReplier< TReq, TRep >::get_write_params_for_related_↩
request (
             rti::pub::WriteParams & write_params,
            const  dds::sub::SampleInfo & related_request_info )  [inline]
```

Gets the write parameters ready to send a reply for a given related request.

You can use this method to modify the parameters before you send a reply using **QueueReplier::send_reply(const TRep&, rti::pub::WriteParams&)** (p. 2099).

Parameters

write_params	<<***out***>> *(*p. *355)* the write params to send the reply.
related_request_info	<<***in***>> *(*p. *355)* The sample info of the releated request.

MT Safety:

SAFE

See also

QueueReplier::send_reply(const TRep&, rti::pub::WriteParams&) (p. 2099)

8.286.3.22 take_requests() [1/2]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TReq >  rti::queuing::QueueReplier< TReq, TRep >::take_requests ( )
[inline]
```

Provides a loaned container to access the existing requests.

See also

QueueConsumer::take_samples() (p. 2071)

8.286.3.23 take_requests() [2/2]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TReq >  rti::queuing::QueueReplier< TReq, TRep >::take_requests (
            int max_count )  [inline]
```

Provides a loaned container to access the existing requests.

See also

QueueConsumer::take_samples(int) (p. 2072)

8.286.3.24 read_requests() [1/2]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TReq >  rti::queuing::QueueReplier< TReq, TRep >::read_requests ( )
[inline]
```

Provides a loaned container from which you can access the existing requests.

This operation is equivalent to **QueueReplier::take_requests()** (p. 2105), except the sample remains in the **Queue↩ Replier** (p. 2094) and can be read or taken again.

8.286.3.25 read_requests() [2/2]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TReq >  rti::queuing::QueueReplier< TReq, TRep >::read_requests (
            int max_count )  [inline]
```

Provides a loaned container from which you can access the existing requests.

This operation is equivalent to **QueueReplier::take_requests(int)** (p. 2105), except the sample remains in the **Queue↩ Replier** (p. 2094) and can be read or taken again.

8.286.3.26 writer()

```
template<typename TReq , typename TRep >
dds::pub::DataWriter< TRep >  rti::queuing::QueueReplier< TReq, TRep >::writer ( ) const  [inline]
```

Retrieves the underlying **dds::pub::DataWriter** (p. 1138) used to send replies.

See also

QueueProducer::writer() (p. 2088)

8.286.3.27 reader()

```
template<typename TReq , typename TRep >
dds::sub::DataReader< TReq >  rti::queuing::QueueReplier< TReq, TRep >::reader ( ) const  [inline]
```

Retrieves the underlying **dds::sub::DataReader** (p. 976) used to receive requests.

See also

QueueConsumer::reader (p. 2070)

8.286.3.28 guid()

```
template<typename TReq , typename TRep >
rti::core::Guid  rti::queuing::QueueReplier< TReq, TRep >::guid ( )  [inline]
```

Returns the GUID of this **QueueReplier** (p. 2094).

The **QueueReplier** (p. 2094)'s GUID is the same GUID of both underlying **QueueProducer** (p. 2080) and **Queue↩ Consumer** (p. 2063).

See also

QueueReplierParams::entity_name

QueueProducer::guid (p. 2089)

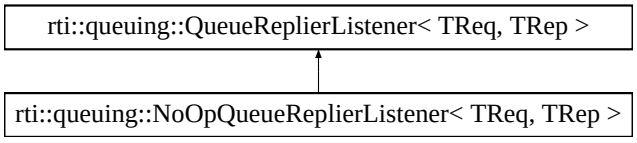
QueueConsumer::guid (p. 2070)

8.287 rti::queuing::QueueReplierListener< TReq, TRep > Class Template Reference

Called when certain events occur in a **QueueReplier** (p. 2094).

```
#include <rti/queuing/QueueReplierListener.hpp>
```

Inheritance diagram for rti::queuing::QueueReplierListener< TReq, TRep >:



Public Member Functions

- virtual void **on_reply_acknowledged** (**QueueReplier**< TReq, TRep > &producer, const **rti::pub::↩ AcknowledgmentInfo** &info)=0

  User callback.
- virtual void **on_request_available** (**QueueReplier**< TReq, TRep > &requester)=0

  User callback.
- virtual void **on_request_shared_reader_queue_matched** (**QueueReplier**< TReq, TRep > &requester, const **dds::core::status::SubscriptionMatchedStatus** &status)=0

  User callback.
- virtual void **on_reply_shared_reader_queue_matched** (**QueueReplier**< TReq, TRep > &requester, const **dds::core::status::PublicationMatchedStatus** &status)=0

  User callback.

8.287.1 Detailed Description

template<typename TReq, typename TRep>
class rti::queuing::QueueReplierListener< TReq, TRep >

Called when certain events occur in a **QueueReplier** (p. 2094).

A **QueueReplier** (p. 2094) listener is a way to implement a callback that will be invoked when certain events happen. It is an optional parameter in **QueueReplierParams** (p. 2109).

See also

QueueProducerListener (p. 2090)

QueueConsumerListener (p. 2075)

QueueReplier::QueueReplier() (p. 2096)

8.287.2 Member Function Documentation

8.287.2.1 on_reply_acknowledged()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::QueueReplierListener< TReq, TRep >::on_reply_acknowledged (
              QueueReplier< TReq, TRep > & producer,
             const  rti::pub::AcknowledgmentInfo & info )  [pure virtual]
```

User callback.

See also

QueueProducerListener::on_sample_acknowledged (p. 2091)

Implemented in **rti::queuing::NoOpQueueReplierListener< TReq, TRep >** (p. 1860).

8.287.2.2 on_request_available()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::QueueReplierListener< TReq, TRep >::on_request_available (
              QueueReplier< TReq, TRep > & requester )  [pure virtual]
```

User callback.

See also

QueueConsumerListener::on_sample_available (p. 2076)

Implemented in **rti::queuing::NoOpQueueReplierListener< TReq, TRep >** (p. 1860).

8.287.2.3 on_request_shared_reader_queue_matched()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::QueueReplierListener< TReq, TRep >::on_request_shared_reader_queue_↩
matched (
             QueueReplier< TReq, TRep > & requester,
            const  dds::core::status::SubscriptionMatchedStatus & status )  [pure virtual]
```

User callback.

See also

QueueConsumerListener::on_shared_reader_queue_matched (p. 2077)

Implemented in **rti::queuing::NoOpQueueReplierListener< TReq, TRep >** (p. 1860).

8.287.2.4 on_reply_shared_reader_queue_matched()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::QueueReplierListener< TReq, TRep >::on_reply_shared_reader_queue_↩
matched (
             QueueReplier< TReq, TRep > & requester,
            const  dds::core::status::PublicationMatchedStatus & status )  [pure virtual]
```

User callback.

See also

QueueProducerListener::on_shared_reader_queue_matched (p. 2091)

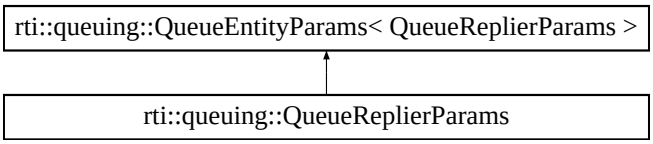
Implemented in **rti::queuing::NoOpQueueReplierListener< TReq, TRep >** (p. 1861).

8.288 rti::queuing::QueueReplierParams Class Reference

Contains the parameters for creating a **QueueReplier** (p. 2094).

```
#include <rti/queuing/QueueParams.hpp>
```

Inheritance diagram for rti::queuing::QueueReplierParams:



Public Member Functions

- **QueueReplierParams** (**dds::domain::DomainParticipant** participant)

  *Creates a **QueueReplierParams** (p. 2109) object with the participant set.*
- **QueueReplierParams** & **enable_availability** (bool enable)

  Indicates whether the availability channel is enabled with the request SharedReaderQueue.
- **QueueReplierParams** & **enable_sample_replication** (bool enable)

  *Causes the **QueueReplier** (p. 2094) to write all the replies with the **dds::sub::SampleInfo::flag** (p. 2293) including the bit **rti::core::SampleFlag::replicate** (p. 2280).*
- **QueueReplierParams** & **enable_wait_for_ack** (bool enable)

  Enables Queuing Service's 'Acknowledgment Management' feature for replies.

8.288.1 Detailed Description

Contains the parameters for creating a **QueueReplier** (p. 2094).

8.288.2 Constructor & Destructor Documentation

8.288.2.1 QueueReplierParams()

```
rti::queuing::QueueReplierParams::QueueReplierParams (
              dds::domain::DomainParticipant participant )  [inline], [explicit]
```

Creates a **QueueReplierParams** (p. 2109) object with the participant set.

The rest of the parameters that can be set in a **QueueReplierParams** (p. 2109) object are optional.

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) a **QueueReplier** (p. 2094) uses to join a domain.

8.288.3 Member Function Documentation

8.288.3.1 enable_availability()

```
QueueReplierParams & rti::queuing::QueueReplierParams::enable_availability (
            bool enable )  [inline]
```

Indicates whether the availability channel is enabled with the request SharedReaderQueue.

See also

QueueConsumerParams::enable_availability.

References **rti::util::network_capture::enable()**.

8.288.3.2 enable_sample_replication()

```
QueueReplierParams & rti::queuing::QueueReplierParams::enable_sample_replication (
            bool enable )  [inline]
```

Causes the **QueueReplier** (p. 2094) to write all the replies with the **dds::sub::SampleInfo::flag** (p. 2293) including the bit **rti::core::SampleFlag::replicate** (p. 2280).

See also

QueueProducerParams::enable_sample_replication (p. 2093)

References **rti::util::network_capture::enable()**.

8.288.3.3 enable_wait_for_ack()

```
QueueReplierParams & rti::queuing::QueueReplierParams::enable_wait_for_ack (
            bool enable )  [inline]
```

Enables Queuing Service's 'Acknowledgment Management' feature for replies.

See also

QueueProducerParams::enable_wait_for_ack

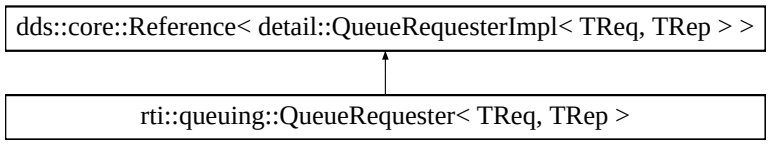
References **rti::util::network_capture::enable()**.

8.289 rti::queuing::QueueRequester< TReq, TRep > Class Template Reference

Allows sending requests and receiving replies.

`#include <rti/queuing/QueueRequester.hpp>`

Inheritance diagram for rti::queuing::QueueRequester< TReq, TRep >:



Public Member Functions

- **QueueRequester** (const **QueueRequesterParams** &params, bool is_enabled=true, **Listener** *requester_listener=NULL, const **rti::core::Guid** &requester_guid= **rti::core::Guid::unknown**())

  Creates a queue requester.
- **QueueRequester** (const **QueueRequesterParams** &params, bool is_enabled, std::shared_ptr< **Listener** > requester_listener, const **rti::core::Guid** &requester_guid= **rti::core::Guid::unknown**())

  *Creates a **QueueRequester** (p. 2111) with parameters.*
- void **listener** (**Listener** *the_listener, const **dds::core::status::StatusMask** &event_mask= **dds::core::status::StatusMask::none**())

  Sets the listener associated with this queue requester.
- **Listener** * **listener** () const

  Returns the listener currently associated with this requester.
- std::shared_ptr< **Listener** > **get_listener** () const

  Gets the listener.
- void **set_listener** (std::shared_ptr< **Listener** > the_listener)

  Sets a listener to be notified of status updates.
- void **enable** ()

  *Enables the **QueueRequester** (p. 2111) to send data, receive data and receive listener notifications.*
- **dds::sub::LoanedSamples**< TRep > **receive_replies** (const **dds::core::Duration** &max_wait)

  Waits for multiple replies and provides a loaned container to access them.
- **dds::sub::LoanedSamples**< TRep > **receive_replies** (int min_count, int max_count, const **dds::core::Duration** &max_wait)

  Waits for multiple replies and provides a loaned container to access them.
- **dds::sub::LoanedSamples**< TRep > **take_replies** ()

  Provides a loaned container from which you can access the existing replies.
- **dds::sub::LoanedSamples**< TRep > **take_replies** (int max_count)

  Provides a loaned container from which you can access the existing replies.
- **dds::sub::LoanedSamples**< TRep > **read_replies** ()

  Provides a loaned container from which you can access the existing replies.
- **dds::sub::LoanedSamples**< TRep > **read_replies** (int max_count)

  Provides a loaned container from which you can access the existing replies for a specific request.
- **dds::sub::LoanedSamples**< TRep > **take_replies** (int max_count, const **rti::core::SampleIdentity** &related_request_id)

  Provides a loaned container to access the existing replies for a specific request.
- **dds::sub::LoanedSamples**< TRep > **take_replies** (const **rti::core::SampleIdentity** &related_request_id)

  Provides a loaned container from which you can access the existing replies for a specific request.
- **dds::sub::LoanedSamples**< TRep > **read_replies** (const **rti::core::SampleIdentity** &related_request_id)

  Provides a loaned container from which you can access the existing replies for a specific request.
- void **send_request** (const TReq &request)

  Sends a request.
- void **send_request** (const TReq &request, **rti::pub::WriteParams** &write_params)

  Sends a request with the specified parameters and gets back metadata information related to the request sent.
- void **acknowledge_reply** (const **dds::sub::SampleInfo** &sample_info, bool is_positive_acknowledgment=true)

  Explicitly acknowledges a single reply.
- void **acknowledge_all** (bool is_positive_acknowledgment=true)

  Acknowledges all previously accessed replies.
- bool **wait_for_acknowledgments** (const **dds::core::Duration** &max_wait)

*Blocks the calling thread until all requests written by this **QueueRequester** (p. 2111) are acknowledged Queuing Service, or until a timeout occurs.*

- bool **wait_for_acknowledgments** (const **rti::core::SampleIdentity** &identity, const **dds::core::Duration** &max_wait)

  Blocks the calling thread until the request identified by identity is acknowledged by Queuing Service, or until a timeout occurs.
- **rti::queuing::QueueConsumer**< TRep > **consumer** () const

  *Retrieves the underlying **QueueConsumer** (p. 2063).*
- **rti::queuing::QueueProducer**< TReq > **producer** () const

  *Retrieves the underlying **QueueProducer** (p. 2080).*
- bool **wait_for_replies** (const **dds::core::Duration** &max_wait)

  Waits for replies to any request.
- bool **wait_for_replies** (int min_count, const **dds::core::Duration** &max_wait)

  Waits for replies to any request.
- bool **wait_for_replies** (int min_count, const **dds::core::Duration** &max_wait, const **rti::core::SampleIdentity** &related_request_id)

  Waits for replies to a specific request.
- void **send_availability** (**ConsumerAvailabilityParams** parameters)

  *Sends the availability status of the **QueueConsumer** (p. 2063) receiving samples from the reply SharedReaderQueue to Queuing Service.*
- bool **has_matching_request_reader_queue** ()

  *Checks whether this **QueueRequester** (p. 2111) has matched with at least one request SharedReaderQueue.*
- bool **has_matching_reply_reader_queue** ()

  *Checks whether this **QueueRequester** (p. 2111) has matched with at least one reply SharedReaderQueue.*
- **dds::pub::DataWriter**< TReq > **writer** () const

  *Retrieves the underlying **dds::pub::DataWriter** (p. 1138) used to send requests.*
- **dds::sub::DataReader**< TRep > **reader** () const

  *Retrieves the underlying **dds::sub::DataReader** (p. 976) used to receive replies.*
- **rti::core::Guid guid** ()

  *Returns the GUID of this **QueueRequester** (p. 2111).*

8.289.1 Detailed Description

template<typename TReq, typename TRep>
class rti::queuing::QueueRequester< TReq, TRep >

Allows sending requests and receiving replies.

A **QueueRequester** (p. 2111) is a component suited for the queuing request-reply use case. A **QueueRequester** (p. 2111) plays the role of a request **QueueProducer** (p. 2080) and reply **QueueConsumer** (p. 2063).

A **QueueRequester** (p. 2111) is the entity that allows sending requests to a a request SharedReaderQueue and receiving replies from a reply SharedReaderQueue.

To communicate with the SharedReaderQueues the **QueueRequester** (p. 2111) creates two topics. The first topic corresponds to the request SharedReaderQueue topic and the second topic to the reply SharedReaderQueue topic.

Valid types for these topics (`TReq` and `TRep`) are: those generated by rtiddsgen, the **DDS built-in types** (p. 50), and **dds::core::xtypes::DynamicData** (p. 1461).

See also

QueueProducer (p. 2080)

QueueConsumer (p. 2063)

Template Parameters

TReq	The data type for the request SharedReaderQueue topic
TRep	The data type for the reply SharedReaderQueue topic

8.289.2 Constructor & Destructor Documentation

8.289.2.1 QueueRequester() [1/2]

```
template<typename TReq , typename TRep >
rti::queuing::QueueRequester< TReq, TRep >::QueueRequester (
            const  QueueRequesterParams & params,
            bool is_enabled = true,
             Listener * requester_listener = NULL,
            const  rti::core::Guid & requester_guid =  rti::core::Guid::unknown() )  [inline],
[explicit]
```

Creates a queue requester.

[DEPRECATED] When using a listener, prefer the constructor that receives a shared_ptr<**Listener** (p. 1637)> instead of a regular Listener* pointer.

8.289.2.2 QueueRequester() [2/2]

```
template<typename TReq , typename TRep >
rti::queuing::QueueRequester< TReq, TRep >::QueueRequester (
            const  QueueRequesterParams & params,
            bool is_enabled,
            std::shared_ptr<  Listener > requester_listener,
            const  rti::core::Guid & requester_guid =  rti::core::Guid::unknown() )  [inline],
[explicit]
```

Creates a **QueueRequester** (p. 2111) with parameters.

Parameters

params	All the parameters that configure this **QueueRequester** (p. 2111). See **QueueRequesterParams** (p. 2128) for the list of mandatory parameters.
is_enabled	Specifies if the **QueueRequester** (p. 2111) is created ready to receive notifications in its listener and send or receive data. If you choose to bind the **QueueRequester** (p. 2111) to a **QueueRequesterListener** (p. 2126) using an **rti::core::ListenerBinder** (p. 1641) you should create the **QueueRequester** (p. 2111) disabled to avoid missing listener notifications. You can later enable the disabled **QueueRequester** (p. 2111) using **QueueRequester::enable** (p. 2116).
requester_listener	A shared_ptr to a **QueueRequesterListener** (p. 2126) object to receive event notifications.
requester_guid	A GUID identifier for the requester. When requester_guid is set to **rti::core::Guid::unknown()** (p. 1597) the Guid identifier is generated automatically. You can retrieve the generated **QueueRequester** (p. 2111) Guid using the **QueueRequester::guid()** (p. 2125) method. To restart a **QueueRequester** (p. 2111) just create a new one with its same GUID.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

QueueRequesterParams (p. 2128)

8.289.3 Member Function Documentation

8.289.3.1 listener() [1/2]

```
template<typename TReq , typename TRep >
void  rti::queuing::QueueRequester< TReq, TRep >::listener (
             Listener * the_listener,
            const  dds::core::status::StatusMask & event_mask =  dds::core::status::StatusMask↩
::none() )  [inline]
```

Sets the listener associated with this queue requester.

[DEPRECATED] The use of **set_listener()** (p. 2116) is recommended. Unlike this function, set_listener receives a shared_ptr which simplifies the management of listener's lifecycle.

Parameters

the_listener	The **QueueRequesterListener** (p. 2126) to set
event_mask	The **dds::core::status::StatusMask** (p. 2375) associated with the listener

8.289.3.2 listener() [2/2]

```
template<typename TReq , typename TRep >
Listener *  rti::queuing::QueueRequester< TReq, TRep >::listener ( ) const  [inline]
```

Returns the listener currently associated with this requester.

[DEPRECATED] Prefer **get_listener()** (p. 2115) instead of this function.

If there is no listener it returns NULL.

8.289.3.3 get_listener()

```
template<typename TReq , typename TRep >
std::shared_ptr< Listener > rti::queuing::QueueRequester< TReq, TRep >::get_listener ( ) const
[inline]
```

Gets the listener.

8.289.3.4 set_listener()

```
template<typename TReq , typename TRep >
void rti::queuing::QueueRequester< TReq, TRep >::set_listener (
            std::shared_ptr< Listener > the_listener ) [inline]
```

Sets a listener to be notified of status updates.

Warning

It's recommended that the listener implementation doesn't hold a permanent reference to this object. If it does, the application needs to manually reset the listener or manually close this object to ensure that there is no cycle that prevents the destruction of these two objects.

Parameters

the_listener	A shared pointer to the listener to receive updates or `nullptr` to reset the current listener and stop receiving updates.

8.289.3.5 enable()

```
template<typename TReq , typename TRep >
void rti::queuing::QueueRequester< TReq, TRep >::enable ( ) [inline]
```

Enables the **QueueRequester** (p. 2111) to send data, receive data and receive listener notifications.

If you create the **QueueRequester** (p. 2111) disabled you can enable it using this mehtod. To avoid missing listener notifciation when you use an **rti::core::ListenerBinder** (p. 1641) to bind the **QueueRequester** (p. 2111) to a **Queue↩ RequesterListener** (p. 2126) use **QueueRequester::enable** (p. 2116) to enable the **QueueRequester** (p. 2111) after it is bound to the listener.

8.289.3.6 receive_replies() [1/2]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TRep >  rti::queuing::QueueRequester< TReq, TRep >::receive_replies (
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for multiple replies and provides a loaned container to access them.

See also

QueueConsumer::receive_samples(const dds::core::Duration&) (p. 2071)

QueueRequester::wait_for_replies(int, const dds::core::Duration&) (p. 2123)

QueueRequester::take_replies(int) (p. 2117)

8.289.3.7 receive_replies() [2/2]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TRep >  rti::queuing::QueueRequester< TReq, TRep >::receive_replies (
            int min_count,
            int max_count,
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for multiple replies and provides a loaned container to access them.

See also

QueueConsumer::receive_samples(int, int, const dds::core::Duration&) (p. 2071)

QueueRequester::wait_for_replies(int, const dds::core::Duration&) (p. 2123)

QueueRequester::take_replies(int) (p. 2117)

8.289.3.8 take_replies() [1/4]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TRep >  rti::queuing::QueueRequester< TReq, TRep >::take_replies ( )
[inline]
```

Provides a loaned container from which you can access the existing replies.

See also

QueueConsumer::take_samples() (p. 2071)

8.289.3.9 take_replies() [2/4]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TRep >  rti::queuing::QueueRequester< TReq, TRep >::take_replies (
            int max_count )  [inline]
```

Provides a loaned container from which you can access the existing replies.

See also

QueueConsumer::take_samples(int) (p. 2072)

8.289.3.10 read_replies() [1/3]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TRep >  rti::queuing::QueueRequester< TReq, TRep >::read_replies ( )
[inline]
```

Provides a loaned container from which you can access the existing replies.

This operation is equivalent to **QueueRequester::take_replies()** (p. 2117), except the sample remains in the **Queue↩ Requester** (p. 2111) and can be read or taken again.

8.289.3.11 read_replies() [2/3]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TRep >  rti::queuing::QueueRequester< TReq, TRep >::read_replies (
            int max_count )  [inline]
```

Provides a loaned container from which you can access the existing replies for a specific request.

This operation is equivalent to **QueueRequester::take_replies(int, const rti::core::SampleIdentity&)** (p. 2118), except the sample remains in the **QueueRequester** (p. 2111) and can be read or taken again.

8.289.3.12 take_replies() [3/4]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TRep >  rti::queuing::QueueRequester< TReq, TRep >::take_replies (
            int max_count,
            const  rti::core::SampleIdentity & related_request_id )  [inline]
```

Provides a loaned container to access the existing replies for a specific request.

This operation is analogous to **QueueRequester::take_replies(int)** (p. 2117), except the replies this operation provides correspond to a specific request.

Parameters

max_count	The maximum number of samples that are taken at a time. The special value **dds::core::LENGTH_UNLIMITED** (p. 438) may be used.
related_request_id	The identity of a request previously sent by this **QueueRequester** (p. 2111)

MT Safety:

SAFE

See also

QueueRequester::take_replies(int) (p. 2117)

dds::sub::LoanedSamples (p. 1663)

8.289.3.13 take_replies() [4/4]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TRep >  rti::queuing::QueueRequester< TReq, TRep >::take_replies (
            const  rti::core::SampleIdentity & related_request_id )  [inline]
```

Provides a loaned container from which you can access the existing replies for a specific request.

This operation is equivalent to using **QueueRequester::take_replies(int, const rti::core::SampleIdentity&)** (p. 2118) with max_count=**dds::core::LENGTH_UNLIMITED** (p. 438)

MT Safety:

SAFE

See also

dds::sub::LoanedSamples (p. 1663)

QueueRequester::take_replies(int, const rti::core::SampleIdentity&) (p. 2118)

8.289.3.14 read_replies() [3/3]

```
template<typename TReq , typename TRep >
dds::sub::LoanedSamples< TRep >  rti::queuing::QueueRequester< TReq, TRep >::read_replies (
            const  rti::core::SampleIdentity & related_request_id )  [inline]
```

Provides a loaned container from which you can access the existing replies for a specific request.

This operation is equivalent to **QueueRequester::take_replies(int, const rti::core::SampleIdentity&)** (p. 2118), except the sample remains in the **QueueRequester** (p. 2111) and can be read or taken again.

8.289.3.15 send_request() [1/2]

```
template<typename TReq , typename TRep >
void  rti::queuing::QueueRequester< TReq, TRep >::send_request (
             const TReq & request )  [inline]
```

Sends a request.

. If a future reply needs to be correlated to exactly this request, use QueueRequester::send_request_w_writesample.

Parameters

request	The request to be sent

Exceptions

One	of the **Standard Exceptions** (p. 428)

MT Safety:

SAFE

See also

QueueProducer::send_sample(const T&, rti::pub::WriteParams&) (p. 2085)

8.289.3.16 send_request() [2/2]

```
template<typename TReq , typename TRep >
void  rti::queuing::QueueRequester< TReq, TRep >::send_request (
             const TReq & request,
              rti::pub::WriteParams & write_params )  [inline]
```

Sends a request with the specified parameters and gets back metadata information related to the request sent.

After calling this operation, write_params contains valid metadata information associated with the request sent.

The metadata field **rti::pub::WriteParams::source_guid** (p. 2650) of the write_params in the request is always set to this **QueueProducer** (p. 2080)'s GUID.

Parameters

request	<<***in***>> *(p. 355)* the request sample to send.
write_params	<<***inout***>> *(p. 355)* Parameters used to send the request. When this call ends successfully, `write_params` contains valid metadata information associated with the request sent.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **NoMatchingQueueException** (p. 1835) if by the time this operation is called there is not a single matching SharedReaderQueue.

MT Safety:

SAFE

See also

QueueProducer::send_sample(const T&) (p. 2085)

QueueProducer::send_sample(const T&, rti::pub::WriteParams&) (p. 2085)

QueueRequester::wait_for_replies(int, const dds::core::Duration&, const rti::core::SampleIdentity&) (p. 2123)

QueueRequester::take_replies(int, const rti::core::SampleIdentity&) (p. 2118)

QueueRequester::take_replies(const rti::core::SampleIdentity&) (p. 2119)

8.289.3.17 acknowledge_reply()

```
template<typename TReq , typename TRep >
void  rti::queuing::QueueRequester< TReq, TRep >::acknowledge_reply (
            const  dds::sub::SampleInfo & sample_info,
            bool is_positive_acknowledgment = true )  [inline]
```

Explicitly acknowledges a single reply.

See also

QueueConsumer::acknowledge_sample(const dds::sub::SampleInfo&, bool) (p. 2069)

QueueRequester::acknowledge_all(bool) (p. 2121)

8.289.3.18 acknowledge_all()

```
template<typename TReq , typename TRep >
void  rti::queuing::QueueRequester< TReq, TRep >::acknowledge_all (
            bool is_positive_acknowledgment = true )  [inline]
```

Acknowledges all previously accessed replies.

See also

QueueConsumer::acknowledge_all(bool) (p. 2069)

QueueRequester::acknowledge_reply(const dds::sub::SampleInfo&, bool) (p. 2121)

8.289.3.19 wait_for_acknowledgments() [1/2]

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueRequester< TReq, TRep >::wait_for_acknowledgments (
            const  dds::core::Duration & max_wait )  [inline]
```

Blocks the calling thread until all requests written by this **QueueRequester** (p. 2111) are acknowledged Queuing Service, or until a timeout occurs.

See also

QueueProducer::wait_for_acknowledgments(const dds::core::Duration&) (p. 2086)

QueueRequester::send_request(const TReq&) (p. 2119)

QueueRequesterListener::on_request_acknowledged (p. 2127)

8.289.3.20 wait_for_acknowledgments() [2/2]

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueRequester< TReq, TRep >::wait_for_acknowledgments (
            const  rti::core::SampleIdentity & identity,
            const  dds::core::Duration & max_wait )  [inline]
```

Blocks the calling thread until the request identified by identity is acknowledged by Queuing Service, or until a timeout occurs.

See also

QueueProducer::wait_for_acknowledgments(const rti::core::SampleIdentity&, const dds::core::↩ Duration&) (p. 2087)

QueueRequester::send_request(const TReq&) (p. 2119)

QueueRequesterListener::on_request_acknowledged (p. 2127)

8.289.3.21 consumer()

```
template<typename TReq , typename TRep >
rti::queuing::QueueConsumer< TRep >  rti::queuing::QueueRequester< TReq, TRep >::consumer ( )
const  [inline]
```

Retrieves the underlying **QueueConsumer** (p. 2063).

8.289.3.22 producer()

```
template<typename TReq , typename TRep >
rti::queuing::QueueProducer< TReq >  rti::queuing::QueueRequester< TReq, TRep >::producer ( )
const  [inline]
```

Retrieves the underlying **QueueProducer** (p. 2080).

8.289.3.23 wait_for_replies() [1/3]

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueRequester< TReq, TRep >::wait_for_replies (
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for replies to any request.

See also

QueueConsumer::wait_for_samples(int, const dds::core::Duration&) (p. 2073)

8.289.3.24 wait_for_replies() [2/3]

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueRequester< TReq, TRep >::wait_for_replies (
            int min_count,
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for replies to any request.

See also

QueueConsumer::wait_for_samples(int, const dds::core::Duration&) (p. 2073)

8.289.3.25 wait_for_replies() [3/3]

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueRequester< TReq, TRep >::wait_for_replies (
            int min_count,
            const  dds::core::Duration & max_wait,
            const  rti::core::SampleIdentity & related_request_id )  [inline]
```

Waits for replies to a specific request.

This operation is analogous to **QueueRequester::wait_for_replies(int, const dds::core::Duration&)** (p. 2123), except this operation waits for replies for a specific request.

Parameters

min_count	Minimum number of replies for the related request that need to be available for this operation to unblock.
max_wait	Maximum wait time after which this operation unblocks, regardless of how many replies are available.
related_request_id	The identity of a request previously sent by this **QueueRequester** (p. 2111)

Returns

true if at least min_count replies for the related request were available before max_wait elapsed, or false otherwise.

Exceptions

One	of the **Standard Exceptions** (p. 428)

MT Safety:

SAFE

See also

QueueRequester::wait_for_replies(int, const dds::core::Duration&) (p. 2123)

8.289.3.26 send_availability()

```
template<typename TReq , typename TRep >
void  rti::queuing::QueueRequester< TReq, TRep >::send_availability (
            ConsumerAvailabilityParams parameters )  [inline]
```

Sends the availability status of the **QueueConsumer** (p. 2063) receiving samples from the reply SharedReaderQueue to Queuing Service.

See also

QueueConsumer::send_availability(ConsumerAvailabilityParams) (p. 2074)

QueueRequesterParams::enable_availability

8.289.3.27 has_matching_request_reader_queue()

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueRequester< TReq, TRep >::has_matching_request_reader_queue ( )  [inline]
```

Checks whether this **QueueRequester** (p. 2111) has matched with at least one request SharedReaderQueue.

See also

QueueProducer::has_matching_reader_queue() (p. 2089)

8.289.3.28 has_matching_reply_reader_queue()

```
template<typename TReq , typename TRep >
bool  rti::queuing::QueueRequester< TReq, TRep >::has_matching_reply_reader_queue ( )  [inline]
```

Checks whether this **QueueRequester** (p. 2111) has matched with at least one reply SharedReaderQueue.

See also

QueueConsumer::has_matching_reader_queue() (p. 2075)

8.289.3.29 writer()

```
template<typename TReq , typename TRep >
dds::pub::DataWriter< TReq >  rti::queuing::QueueRequester< TReq, TRep >::writer ( ) const  [inline]
```

Retrieves the underlying **dds::pub::DataWriter** (p. 1138) used to send requests.

See also

QueueProducer::writer() (p. 2088)

8.289.3.30 reader()

```
template<typename TReq , typename TRep >
dds::sub::DataReader< TRep >  rti::queuing::QueueRequester< TReq, TRep >::reader ( ) const  [inline]
```

Retrieves the underlying **dds::sub::DataReader** (p. 976) used to receive replies.

See also

QueueConsumer::reader (p. 2070)

8.289.3.31 guid()

```
template<typename TReq , typename TRep >
rti::core::Guid rti::queuing::QueueRequester< TReq, TRep >::guid ( ) [inline]
```

Returns the GUID of this **QueueRequester** (p. 2111).

The **QueueRequester** (p. 2111)'s GUID is the same GUID of both the underlying **QueueProducer** (p. 2080) and **QueueConsumer** (p. 2063).

The GUID of the **QueueRequester** (p. 2111) is also used by Queuing Service to correlate the replies the **QueueRequester** (p. 2111) produces. Replies are only delivered to the **QueueRequester** (p. 2111) that generated the related request.

See also

QueueRequesterParams::entity_name

QueueProducer::guid (p. 2089)

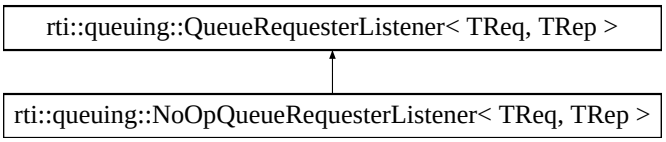
QueueConsumer::guid (p. 2070)

8.290 rti::queuing::QueueRequesterListener< TReq, TRep > Class Template Reference

Called when certain events occur in a **QueueRequester** (p. 2111).

```
#include <rti/queuing/QueueRequesterListener.hpp>
```

Inheritance diagram for rti::queuing::QueueRequesterListener< TReq, TRep >:



Public Member Functions

- virtual void **on_request_acknowledged** (**QueueRequester**< TReq, TRep > &requester, const **rti::pub::AcknowledgmentInfo** &info)=0

  User callback.
- virtual void **on_reply_available** (**QueueRequester**< TReq, TRep > &requester)=0

  User callback.
- virtual void **on_request_shared_reader_queue_matched** (**QueueRequester**< TReq, TRep > &requester, const **dds::core::status::PublicationMatchedStatus** &status)=0

  User callback.
- virtual void **on_reply_shared_reader_queue_matched** (**QueueRequester**< TReq, TRep > &requester, const **dds::core::status::SubscriptionMatchedStatus** &status)=0

  User callback.

8.290.1 Detailed Description

template<typename TReq, typename TRep>
class rti::queuing::QueueRequesterListener< TReq, TRep >

Called when certain events occur in a **QueueRequester** (p. 2111).

A **QueueRequester** (p. 2111) listener is a way to implement a callback that will be invoked when certain events happen. It is an optional parameter in **QueueRequesterParams** (p. 2128).

See also

QueueProducerListener (p. 2090)

QueueConsumerListener (p. 2075)

QueueRequester::QueueRequester() (p. 2114)

8.290.2 Member Function Documentation

8.290.2.1 on_request_acknowledged()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::QueueRequesterListener< TReq, TRep >::on_request_acknowledged (
              QueueRequester< TReq, TRep > & requester,
             const  rti::pub::AcknowledgmentInfo & info )  [pure virtual]
```

User callback.

See also

QueueProducerListener::on_sample_acknowledged (p. 2091)

Implemented in **rti::queuing::NoOpQueueRequesterListener< TReq, TRep >** (p. 1862).

8.290.2.2 on_reply_available()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::QueueRequesterListener< TReq, TRep >::on_reply_available (
              QueueRequester< TReq, TRep > & requester )  [pure virtual]
```

User callback.

See also

QueueConsumerListener::on_sample_available (p. 2076)

Implemented in **rti::queuing::NoOpQueueRequesterListener< TReq, TRep >** (p. 1862).

8.290.2.3 on_request_shared_reader_queue_matched()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::QueueRequesterListener< TReq, TRep >::on_request_shared_reader_↩
queue_matched (
              QueueRequester< TReq, TRep > & requester,
             const  dds::core::status::PublicationMatchedStatus & status )  [pure virtual]
```

User callback.

See also

QueueProducerListener::on_shared_reader_queue_matched (p. 2091)

Implemented in **rti::queuing::NoOpQueueRequesterListener< TReq, TRep >** (p. 1863).

8.290.2.4 on_reply_shared_reader_queue_matched()

```
template<typename TReq , typename TRep >
virtual void  rti::queuing::QueueRequesterListener< TReq, TRep >::on_reply_shared_reader_queue_↩
matched (
              QueueRequester< TReq, TRep > & requester,
             const  dds::core::status::SubscriptionMatchedStatus & status )  [pure virtual]
```

User callback.

See also

QueueConsumerListener::on_shared_reader_queue_matched (p. 2077)

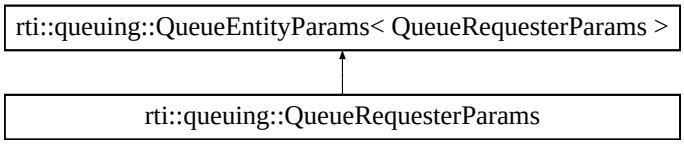
Implemented in **rti::queuing::NoOpQueueRequesterListener< TReq, TRep >** (p. 1863).

8.291 rti::queuing::QueueRequesterParams Class Reference

Contains the parameters for creating a **QueueRequester** (p. 2111).

```
#include <rti/queuing/QueueParams.hpp>
```

Inheritance diagram for rti::queuing::QueueRequesterParams:



Public Member Functions

- **QueueRequesterParams** (**dds::domain::DomainParticipant** participant)

  *Creates a **QueueRequesterParams** (p. 2128) object with the participant set.*
- **QueueRequesterParams** & **enable_availability** (bool enable)

  Indicates whether the availability channel is enabled with the reply SharedReaderQueue.
- **QueueRequesterParams** & **enable_sample_replication** (bool enable)

  *Causes the **QueueRequester** (p. 2111) to write all the requests with the **dds::sub::SampleInfo::flag** (p. 2293) including the bit **rti::core::SampleFlag::replicate** (p. 2280).*
- **QueueRequesterParams** & **enable_wait_for_ack** (bool enable)

  Enables Queuing Service's 'Acknowledgment Management' feature for requests.

8.291.1 Detailed Description

Contains the parameters for creating a **QueueRequester** (p. 2111).

8.291.2 Constructor & Destructor Documentation

8.291.2.1 QueueRequesterParams()

```
rti::queuing::QueueRequesterParams::QueueRequesterParams (
              dds::domain::DomainParticipant participant )  [inline], [explicit]
```

Creates a **QueueRequesterParams** (p. 2128) object with the participant set.

The rest of the parameters that can be set in a **QueueRequesterParams** (p. 2128) object are optional.

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) a **QueueRequester** (p. 2111) uses to join a domain.

8.291.3 Member Function Documentation

8.291.3.1 enable_availability()

```
QueueRequesterParams & rti::queuing::QueueRequesterParams::enable_availability (
            bool enable )  [inline]
```

Indicates whether the availability channel is enabled with the reply SharedReaderQueue.

See also

QueueConsumerParams::enable_availability.

References **rti::util::network_capture::enable()**.

8.291.3.2 enable_sample_replication()

```
QueueRequesterParams & rti::queuing::QueueRequesterParams::enable_sample_replication (
            bool enable )  [inline]
```

Causes the **QueueRequester** (p. 2111) to write all the requests with the **dds::sub::SampleInfo::flag** (p. 2293) including the bit **rti::core::SampleFlag::replicate** (p. 2280).

See also

QueueProducerParams::enable_sample_replication (p. 2093)

References **rti::util::network_capture::enable()**.

8.291.3.3 enable_wait_for_ack()

```
QueueRequesterParams & rti::queuing::QueueRequesterParams::enable_wait_for_ack (
            bool enable )  [inline]
```

Enables Queuing Service's 'Acknowledgment Management' feature for requests.

See also

QueueProducerParams::enable_wait_for_ack

References **rti::util::network_capture::enable()**.

8.292 dds::sub::Rank Class Reference

<<***value-type***>> *(p. 350)* Contains the sample and generation ranks of a data-sample

```
#include <TRank.hpp>
```

Public Member Functions

- **Rank** ()

  *Create a default **Rank** (p. 2130) object.*
- **Rank** (int32_t sample_rank, int32_t generation_rank, int32_t absolute_generation_rank)

  *Create a **Rank** (p. 2130) object with the provided sample_rank, generation_rank, and absolute_generation_rank.*
- int32_t **sample** () const

  Get the sample rank of the sample.
- int32_t **generation** () const

  Get the generation rank of the sample.
- int32_t **absolute_generation** () const

  Get the absolute generation rank of the sample.

8.292.1 Detailed Description

<<***value-type***>> *(p. 350)* Contains the sample and generation ranks of a data-sample

See also

dds::sub::SampleInfo::rank() (p. 2289)

8.292.2 Constructor & Destructor Documentation

8.292.2.1 Rank() [1/2]

```
dds::sub::Rank::Rank ( )  [inline]
```

Create a default **Rank** (p. 2130) object.

8.292.2.2 Rank() [2/2]

```
dds::sub::Rank::Rank (
            int32_t sample_rank,
            int32_t generation_rank,
            int32_t absolute_generation_rank )  [inline]
```

Create a **Rank** (p. 2130) object with the provided sample_rank, generation_rank, and absolute_generation_rank.

8.292.3 Member Function Documentation

8.292.3.1 sample()

```
int32_t dds::sub::Rank::sample ( ) const  [inline]
```

Get the sample rank of the sample.

The sample rank of the sample.

Indicates the number of samples related to the same instance that follow in the collection returned by `read` or `take`.

See also

Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank (p. 2288)
Interpretation of the SampleInfo counters and ranks (p. 2288)

8.292.3.2 generation()

```
int32_t dds::sub::Rank::generation ( ) const  [inline]
```

Get the generation rank of the sample.

The generation rank of the sample.

Indicates the generation difference (number of times the instance was NOT_ALIVE and become alive again) between the time the sample was received and the time the most recent sample in the collection related to the same instance was received.

See also

Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank (p. 2288)
Interpretation of the SampleInfo counters and ranks (p. 2288)

8.292.3.3 absolute_generation()

```
int32_t dds::sub::Rank::absolute_generation ( ) const  [inline]
```

Get the absolute generation rank of the sample.

The absolute generation rank of the sample.

Indicates the generation difference (number of times the instance was disposed and become alive again) between the time the sample was received, and the time the most recent sample (which may not be in the returned collection) related to the same instance was received.

See also

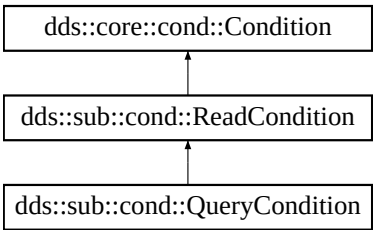
Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank (p. 2288)
Interpretation of the SampleInfo counters and ranks (p. 2288)

8.293 dds::sub::cond::ReadCondition Class Reference

<<***reference-type***>> *(p. 351)* Condition specifically dedicated to read operations and attached to one **dds::sub::DataReader** (p. 976).

`#include <dds/sub/cond/ReadCondition.hpp>`

Inheritance diagram for dds::sub::cond::ReadCondition:



Public Member Functions

- template<typename T >
  ReadCondition (const **dds::sub::DataReader**< T > &reader, const **dds::sub::status::DataState** &status)

  *Creates a **ReadCondition** (p. 2133).*
- template<typename T , typename Functor >
  ReadCondition (const **dds::sub::DataReader**< T > &reader, const **dds::sub::status::DataState** &status, const Functor &handler)

  *Creates a **ReadCondition** (p. 2133) with a handler.*
- const **dds::sub::status::DataState** **state_filter** () const

  Returns the DataState of this condition.
- const **AnyDataReader** & **data_reader** () const

  *Returns the **DataReader** (p. 976) associated to this condition.*
- void **close** ()

  *Closes the **ReadCondition** (p. 2133).*
- bool **closed** () const

  *Indicates whether this **ReadCondition** (p. 2133) has been closed with **close()** (p. 2136).*

Related Functions

(Note that these are not member functions.)

- dds::sub::cond::detail::ReadCondition **create_read_condition_ex** (const **dds::sub::AnyDataReader** &reader, const **::rti::sub::status::DataStateEx** &status)

  <<***extension***>> *(p. 354) Creates a **ReadCondition** (p. 2133) with the extended DataStateEx.*
- template<typename Functor >
  dds::sub::cond::detail::ReadCondition **create_read_condition_ex** (const **dds::sub::AnyDataReader** &reader, const **::rti::sub::status::DataStateEx** &status, const Functor &handler)

  <<***extension***>> *(p. 354) Creates a **ReadCondition** (p. 2133) with the extended DataStateEx and a handler.*

8.293.1 Detailed Description

<<***reference-type***>> *(p. 351)* Condition specifically dedicated to read operations and attached to one **dds::sub::DataReader** (p. 976).

dds::sub::cond::ReadCondition (p. 2133) objects allow an application to specify the data samples it is interested in (by specifying the desired `sample_states`, `view_states` as well as `instance_states` in **dds::sub::DataReader::read** (p. 996) and **dds::sub::DataReader::take** (p. 997) variants.

This allows RTI Connext to enable the condition only when suitable information is available. They are to be used in conjunction with a WaitSet as normal conditions.

More than one **dds::sub::cond::ReadCondition** (p. 2133) may be attached to the same **dds::sub::DataReader** (p. 976).

Examples

Foo_subscriber.cxx.

8.293.2 Constructor & Destructor Documentation

8.293.2.1 ReadCondition() [1/2]

```
template<typename T >
dds::sub::cond::ReadCondition::ReadCondition (
            const  dds::sub::DataReader< T > & reader,
            const  dds::sub::status::DataState & status )  [inline]
```

Creates a **ReadCondition** (p. 2133).

Template Parameters

T	The topic-type of the **DataReader** (p. 976).

Parameters

reader	The reader associated to the condition. One reader can have multiple ReadConditions but one **ReadCondition** (p. 2133) belongs to exactly one **DataReader** (p. 976).
status	The sample, view and instance states of interest.

See also

rti::sub::cond::create_read_condition_ex

8.293.2.2 ReadCondition() [2/2]

```
template<typename T , typename Functor >
dds::sub::cond::ReadCondition::ReadCondition (
             const  dds::sub::DataReader< T > & reader,
             const  dds::sub::status::DataState & status,
             const Functor & handler )  [inline]
```

Creates a **ReadCondition** (p. 2133) with a handler.

Template Parameters

T	The topic-type of the **DataReader** (p. 976).
Functor	Any type whose instances can be called with a no-argument function call (i.e. `f()`, if `f` is an instance of `Functor`), or with a Condition argument (`f(condition)`) Examples are functions, types that override `operator()`, and lambdas.

Parameters

reader	The reader associated to the condition. One reader can have multiple ReadConditions but one **ReadCondition** (p. 2133) belongs to exactly one **DataReader** (p. 976).
status	The sample, view and instance states of interest.
handler	The handler that **dds::core::cond::WaitSet::dispatch()** (p. 2624) will call when the condition is triggered. This argument is copied.

The following examples creates two ReadConditions with handlers. The first one receives a lambda function with no arguments. The second one receives a function with the condition as an argument.

```
// Example 1: lambda function with no arguments
dds::sub::cond::ReadCondition my_read_condition(
       reader,
       dds::sub::status::DataState::any(),
       []()
       {
           std::cout « "condition has been triggered\n";
       }
);
// Example 2: function with a condition argument
void function_handler(dds::core::cond::Condition c)
{
    // function receiving the condition
    std::cout « "condition has been triggered\n";
}
dds::sub::cond::ReadCondition my_read_condition2(
       reader,
       dds::sub::status::DataState::any(),
       function_handler
);
```

See also

dds::core::cond::WaitSet::dispatch() (p. 2624)

WaitSet wait and dispatch examples (p. 330)

8.293.3 Member Function Documentation

8.293.3.1 state_filter()

```
const dds::sub::status::DataState dds::sub::cond::ReadCondition::state_filter ( ) const [inline]
```

Returns the DataState of this condition.

8.293.3.2 data_reader()

```
const AnyDataReader & dds::sub::cond::ReadCondition::data_reader ( ) const [inline]
```

Returns the **DataReader** (p. 976) associated to this condition.

To obtain the actual typed **DataReader** (p. 976), see **AnyDataReader::get()** (p. 810)

8.293.3.3 close()

```
void dds::sub::cond::ReadCondition::close ( ) [inline]
```

Closes the **ReadCondition** (p. 2133).

This is a explicit deletion, which otherwise would occur when all references to this **ReadCondition** (p. 2133) go out of scope.

Precondition

The **ReadCondition** (p. 2133) must not be attached to any Waitsets.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	If the **ReadCondition** (p. 2133) is attached to a Waitset when it is closed.

8.293.3.4 closed()

```
bool dds::sub::cond::ReadCondition::closed ( ) const [inline]
```

Indicates whether this **ReadCondition** (p. 2133) has been closed with **close()** (p. 2136).

8.293.4 Friends And Related Function Documentation

8.293.4.1 create_read_condition_ex() [1/2]

```
dds::sub::cond::detail::ReadCondition create_read_condition_ex (
            const  dds::sub::AnyDataReader & reader,
            const  ::rti::sub::status::DataStateEx & status )  [related]
```

<<***extension***>> *(p. 354)* Creates a **ReadCondition** (p. 2133) with the extended DataStateEx.

Note

This is a standalone function in the namespace rti::sub::cond

The usual way to create a **dds::sub::cond::ReadCondition** (p. 2133) is the constructor that receives a **dds::sub↩ ::status::DataState** (p. 1118). This extension function allows using **rti::sub::status::DataStateEx** (p. 1126), which includes additional state masks.

References **dds::sub::condition()**.

8.293.4.2 create_read_condition_ex() [2/2]

```
template<typename Functor >
dds::sub::cond::detail::ReadCondition create_read_condition_ex (
            const  dds::sub::AnyDataReader & reader,
            const  ::rti::sub::status::DataStateEx & status,
            const Functor & handler )  [related]
```

<<***extension***>> *(p. 354)* Creates a **ReadCondition** (p. 2133) with the extended DataStateEx and a handler.

Note

This is a standalone function in the namespace rti::sub::cond

The usual way to create a **dds::sub::cond::ReadCondition** (p. 2133) is the constructor that receives a **dds::sub↩ ::status::DataState** (p. 1118). This extension function allows using **rti::sub::status::DataStateEx** (p. 1126), which includes additional state masks.

References **dds::sub::condition()**.

8.294 dds::core::policy::ReaderDataLifecycle Class Reference

Controls how a DataReader manages the lifecycle of the data that it has received.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **ReaderDataLifecycle** ()

  Creates the default policy.
- **ReaderDataLifecycle** (const **dds::core::Duration** &the_nowriter_delay, const **dds::core::Duration** &the_↩ disposed_samples_delay)

  Creates an instance with the specified nowriter and disposed-samples purge delays.
- **ReaderDataLifecycle** & **autopurge_nowriter_samples_delay** (const **dds::core::Duration** &duration)

  Minimum duration for which the DataReader will maintain samples for an instance once its `instance_state` becomes InstanceStateKind::NOT_ALIVE_NO_WRITERS.
- **dds::core::Duration autopurge_nowriter_samples_delay** () const

  Getter (see setter with the same name)
- **ReaderDataLifecycle** & **autopurge_disposed_samples_delay** (const **dds::core::Duration** &duration)

  Minimum duration for which the DataReader will maintain samples for an instance once its `instance_state` becomes InstanceStateKind::NOT_ALIVE_DISPOSED.
- **dds::core::Duration autopurge_disposed_samples_delay** () const

  Getter (see setter with the same name)
- **dds::core::policy::ReaderDataLifecycle** & **autopurge_disposed_instances_delay** (const **dds::core::↩ Duration** &duration)

  *<<**extension**>> (p. 116) Minimum duration for which the DataReader will maintain an instance once its `instance↩ _state` becomes InstanceStateKind::NOT_ALIVE_DISPOSED.*
- **dds::core::Duration autopurge_disposed_instances_delay** () const

  *<<**extension**>> (p. 116) Getter (see setter with the same name)*
- **dds::core::policy::ReaderDataLifecycle** & **autopurge_nowriter_instances_delay** (const **dds::core::↩ Duration** &duration)

  *<<**extension**>> (p. 116) Minimum duration for which the DataReader will maintain an instance once its `instance↩ _state` becomes InstanceStateKind::NOT_ALIVE_NO_WRITERS.*
- **dds::core::Duration autopurge_nowriter_instances_delay** () const

  *<<**extension**>> (p. 116) Getter (see setter with the same name)*

Static Public Member Functions

- static **ReaderDataLifecycle NoAutoPurge** ()

  Returns a policy where all purge delays are disabled (the default)
- static **ReaderDataLifecycle AutoPurgeDisposedSamples** (const **dds::core::Duration** &duration)

  Returns a policy where only the disposed-samples purge delay is enabled with a specified duration.
- static **ReaderDataLifecycle AutoPurgeNoWriterSamples** (const **dds::core::Duration** &d)

  Returns a policy where only the no-writer purge delay is enabled with a specified duration.

8.294.1 Detailed Description

Controls how a DataReader manages the lifecycle of the data that it has received.

When a DataReader receives data, it is stored in a receive queue for the DataReader. The user application may either take the data from the queue or leave it there.

This QoS policy controls whether or not RTI Connext will automatically remove data from the receive queue (so that user applications cannot access it afterwards) when it detects that there are no more DataWriters alive for that data. It specifies how long a **dds::sub::DataReader** (p. 976) must retain information regarding instances that have the instance_state **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617).

Note: This policy is not concerned with keeping reliable reader state or discovery information.

The **dds::sub::DataReader** (p. 976) internally maintains the samples that have not been "taken" by the application, subject to the constraints imposed by other QoS policies such as **dds::core::policy::History** (p. 1602) and **dds::core::policy::ResourceLimits** (p. 2203).

The **dds::sub::DataReader** (p. 976) also maintains information regarding the identity, `view_state` and `instance_state` of data instances even after all samples have been taken. This is needed to properly compute the states when future samples arrive.

Under normal circumstances the **dds::sub::DataReader** (p. 976) can only reclaim all resources for instances for which there are no writers and for which all samples have been 'taken'. The last sample the **dds::sub::DataReader** (p. 976) will have taken for that instance will have an `instance_state` of either **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) or **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) depending on whether or not the last writer that had ownership of the instance disposed it.

In the absence of **READER_DATA_LIFECYCLE** (p. 543), this behavior could cause problems if the application forgets to take those samples. "Untaken" samples will prevent the **dds::sub::DataReader** (p. 976) from reclaiming the resources and they would remain in the **dds::sub::DataReader** (p. 976) indefinitely.

A DataReader can also reclaim all resources for instances that have an instance state of **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) and for which all DDS samples have been 'taken'. DataReaders will only reclaim resources in this situation when the autopurge_disposed_instances_delay has been set to zero.

For keyed Topics, the consideration of removing data samples from the receive queue is done on a per instance (key) basis. Thus when RTI Connext detects that there are no longer DataWriters alive for a certain key value of a Topic (an instance of the Topic), it can be configured to remove all data samples for that instance (key).

Entity:

dds::sub::DataReader (p. 976)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **YES** (p. 514)

8.294.2 Constructor & Destructor Documentation

8.294.2.1 ReaderDataLifecycle() [1/2]

```
dds::core::policy::ReaderDataLifecycle::ReaderDataLifecycle ( )  [inline]
```

Creates the default policy.

8.294.2.2 ReaderDataLifecycle() [2/2]

```
dds::core::policy::ReaderDataLifecycle::ReaderDataLifecycle (
            const  dds::core::Duration & the_nowriter_delay,
            const  dds::core::Duration & the_disposed_samples_delay )  [inline]
```

Creates an instance with the specified nowriter and disposed-samples purge delays.

8.294.3 Member Function Documentation

8.294.3.1 autopurge_nowriter_samples_delay() [1/2]

```
ReaderDataLifecycle & dds::core::policy::ReaderDataLifecycle::autopurge_nowriter_samples_delay (
            const  dds::core::Duration & duration )  [inline]
```

Minimum duration for which the DataReader will maintain samples for an instance once its `instance_state` becomes InstanceStateKind::NOT_ALIVE_NO_WRITERS.

At some point after this time elapses, the **dds::sub::DataReader** (p. 976) will purge all samples for the instance; any "untaken" samples will also be dropped.

[default] dds::core::Duration::infinite() (p. 1451)

[range] [1 nanosec, 1 year] or **dds::core::Duration::infinite()** (p. 1451)

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193), **Sample Memory Management** (p. 230), **Data Availability** (p. 167)

8.294.3.2 autopurge_nowriter_samples_delay() [2/2]

```
dds::core::Duration dds::core::policy::ReaderDataLifecycle::autopurge_nowriter_samples_delay ( )
const  [inline]
```

Getter (see setter with the same name)

8.294.3.3 autopurge_disposed_samples_delay() [1/2]

```
ReaderDataLifecycle & dds::core::policy::ReaderDataLifecycle::autopurge_disposed_samples_delay (
            const  dds::core::Duration & duration )  [inline]
```

Minimum duration for which the DataReader will maintain samples for an instance once its `instance_state` becomes InstanceStateKind::NOT_ALIVE_DISPOSED.

After this time elapses, the **dds::sub::DataReader** (p. 976) will purge all samples for the instance even if they have not been read by the application. This purge is done lazily when space is needed for other samples or instances.

[default] dds::core::Duration::infinite() (p. 1451)

[range] [1 nanosec, 1 year] or **dds::core::Duration::infinite()** (p. 1451)

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193), **Sample Memory Management** (p. 230), **Data Availability** (p. 167)

8.294.3.4 autopurge_disposed_samples_delay() [2/2]

```
dds::core::Duration dds::core::policy::ReaderDataLifecycle::autopurge_disposed_samples_delay ( )
const  [inline]
```

Getter (see setter with the same name)

8.294.3.5 NoAutoPurge()

```
static  ReaderDataLifecycle dds::core::policy::ReaderDataLifecycle::NoAutoPurge ( )  [inline],
[static]
```

Returns a policy where all purge delays are disabled (the default)

[DEPRECATED] Use the default constructor instead

8.294.3.6 AutoPurgeDisposedSamples()

```
static ReaderDataLifecycle dds::core::policy::ReaderDataLifecycle::AutoPurgeDisposedSamples (
            const dds::core::Duration & duration ) [inline], [static]
```

Returns a policy where only the disposed-samples purge delay is enabled with a specified duration.

8.294.3.7 AutoPurgeNoWriterSamples()

```
static ReaderDataLifecycle dds::core::policy::ReaderDataLifecycle::AutoPurgeNoWriterSamples (
            const dds::core::Duration & d ) [inline], [static]
```

Returns a policy where only the no-writer purge delay is enabled with a specified duration.

8.294.3.8 autopurge_disposed_instances_delay() [1/2]

```
dds::core::policy::ReaderDataLifecycle & autopurge_disposed_instances_delay (
            const dds::core::Duration & duration )
```

<<***extension***>> *(p. 116)* Minimum duration for which the DataReader will maintain an instance once its `instance_state` becomes InstanceStateKind::NOT_ALIVE_DISPOSED.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

After this time elapses, when the last sample for the disposed instance is taken, the **dds::sub::DataReader** (p. 976) will keep only the minimum state about the instance.

If you do not want to keep this minimum state after the delay period, also set keep_minimum_state_for_instances to FALSE in **rti::core::policy::DataReaderResourceLimits** (p. 1081).

The only currently supported values are **dds::core::Duration::zero()** (p. 1450) and **dds::core::Duration::infinite()** (p. 1451). A value of **dds::core::Duration::zero()** (p. 1450) will purge an instance's state immediately after the instance state transitions to **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617), as long as all samples, including the dispose sample, associated with that instance have been 'taken'.

[default] dds::core::Duration::infinite() (p. 1451)

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193), **Sample Memory Management** (p. 230), **Data Availability** (p. 167)

8.294.3.9 autopurge_disposed_instances_delay() [2/2]

```
dds::core::Duration autopurge_disposed_instances_delay ( ) const
```

<<***extension***>> *(*p. *116)* Getter (see setter with the same name)

8.294.3.10 autopurge_nowriter_instances_delay() [1/2]

```
dds::core::policy::ReaderDataLifecycle & autopurge_nowriter_instances_delay (
            const  dds::core::Duration & duration )
```

<<***extension***>> *(*p. *116)* Minimum duration for which the DataReader will maintain an instance once its instance_state becomes InstanceStateKind::NOT_ALIVE_NO_WRITERS.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

An instance will transition to the **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) instance_state when all known writers for the instance have either lost liveliness or have unregistered themselves from the instance. After this time elapses, when the last sample for the instance without writers is taken, the **dds::sub::DataReader** (p. 976) will keep only the minimum state about the instance.

If you do not want to keep this minimum state after the delay period, also set keep_minimum_state_for_instances to FALSE in **rti::core::policy::DataReaderResourceLimits** (p. 1081).

The only currently supported values are **dds::core::Duration::zero()** (p. 1450) and **dds::core::Duration::infinite()** (p. 1451). A value of **dds::core::Duration::zero()** (p. 1450) will purge an instance's state immediately after the instance state transitions to **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617), as long as all samples, including the no_writers sample, associated with that instance have been 'taken'.

[default] dds::core::Duration::zero() (p. 1450)

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193), **Sample Memory Management** (p. 230), **Data Availability** (p. 167)

8.294.3.11 autopurge_nowriter_instances_delay() [2/2]

```
dds::core::Duration autopurge_nowriter_instances_delay ( ) const
```

<<***extension***>> *(*p. *116)* Getter (see setter with the same name)

8.295 dds::sub::ReadModeDummyType Class Reference

`#include <TDataReader.hpp>`

8.295.1 Detailed Description

This type is used by the **read()** (p. 661) and **take()** (p. 662) functions that are meant to be passed to the Manipulator↩ Selector. It makes sure that the user can't pass any function which takes 0 arguments

See also

dds::sub::read(dds::sub::ReadModeDummyType) (p. 661)

dds::sub::take(dds::sub::ReadModeDummyType) (p. 662)

8.296 rti::core::policy::ReceiverPool Class Reference

<<***extension***>> *(p. 116)* Configures threads that RTI Connext uses to receive and process data from the transport modules (such as UDP)

`#include <rti/core/policy/CorePolicy.hpp>`

Public Member Functions

- **ReceiverPool** ()

  Creates the default policy.
- **ReceiverPool** (const **rti::core::ThreadSettings** &the_thread, int32_t the_buffer_size, int32_t the_buffer_↩ alignment)

  Creates an instance with the thread settings, buffer size and buffer alignment configuration.
- **ReceiverPool** & **thread** (const **rti::core::ThreadSettings** &the_thread)

  Configures the receiver pool thread(s)
- const **rti::core::ThreadSettings** & **thread** () const

  Getter (see setter with the same name)
- **rti::core::ThreadSettings** & **thread** ()

  Getter (see setter with the same name)
- **ReceiverPool** & **buffer_size** (int32_t the_buffer_size)

  Sets the length of the buffer used to store the incoming raw data.
- int32_t **buffer_size** () const

  Getter (see setter with the same name)
- **ReceiverPool** & **buffer_alignment** (int32_t the_buffer_alignment)

  [DEPRECATED] *(p. **296**) Sets the receive buffer alignment*
- int32_t **buffer_alignment** () const

  Getter (see setter with the same name)

8.296.1 Detailed Description

<<***extension***>> *(*p. *116)* Configures threads that RTI Connext uses to receive and process data from the transport modules (such as UDP)

This QoS policy is an extension to the DDS standard.

Entity:

dds::domain::DomainParticipant (p. 1326)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

See also

Controlling CPU Core Affinity for RTI Threads (p. 2453)

8.296.2 Usage

This QoS policy sets the thread properties such as priority level and stack size for the threads used by the middleware to receive and process data from transports.

RTI uses a separate receive thread per port per transport plug-in. To force RTI Connext to use a separate thread to process the data for a **dds::sub::DataReader** (p. 976), set a unique port for the **rti::core::policy::TransportUnicast** (p. 2556) or **rti::core::policy::TransportMulticast** (p. 2543) for the **dds::sub::DataReader** (p. 976).

This QoS policy also sets the size of the buffer used to store packets received from a transport. This buffer size will limit the largest single packet of data that a **dds::domain::DomainParticipant** (p. 1326) will accept from a transport. Users will often set this size to the largest packet that any of the transports used by their application will deliver. For many applications, the value 65,536 (64 K) is a good choice; this value is the largest packet that can be sent/received via UDP.

8.296.3 Constructor & Destructor Documentation

8.296.3.1 ReceiverPool() [1/2]

```
rti::core::policy::ReceiverPool::ReceiverPool ( )  [inline]
```

Creates the default policy.

8.296.3.2 ReceiverPool() [2/2]

```
rti::core::policy::ReceiverPool::ReceiverPool (
            const  rti::core::ThreadSettings & the_thread,
            int32_t the_buffer_size,
            int32_t the_buffer_alignment )
```

Creates an instance with the thread settings, buffer size and buffer alignment configuration.

See individual setters.

8.296.4 Member Function Documentation

8.296.4.1 thread() [1/3]

```
ReceiverPool & rti::core::policy::ReceiverPool::thread (
            const  rti::core::ThreadSettings & the_thread )
```

Configures the receiver pool thread(s)

There is at least one receive thread, possibly more.

[default] priority above normal.
The actual value depends on your architecture:

For Windows: 2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] mask **rti::core::ThreadSettingsKindMask::floating_point()** (p. 2456) | **rti::core::ThreadSettingsKind↩ Mask::stdio()** (p. 2456)

Categories

Immutable (p. 243), **Concurrency** (p. 164)

8.296.4.2 thread() [2/3]

```
const  rti::core::ThreadSettings & rti::core::policy::ReceiverPool::thread ( ) const
```

Getter (see setter with the same name)

8.296.4.3 thread() [3/3]

```
rti::core::ThreadSettings & rti::core::policy::ReceiverPool::thread ( )
```

Getter (see setter with the same name)

8.296.4.4 buffer_size() [1/2]

```
ReceiverPool & rti::core::policy::ReceiverPool::buffer_size (
            int32_t the_buffer_size )
```

Sets the length of the buffer used to store the incoming raw data.

The receive buffer is used by the receive thread to store the raw data that arrives over the transports in non-zero-copy transports.

Zero-copy transports do not copy their data into the buffer provided by the receive thread. Instead, they provide the receive thread data in buffers allocated by the transports themselves. Only the shared memory built-in transport (SHMEM) supports zero-copy.

buffer_size must always be at least as large as the maximum **NDDS_Transport_Property_t::message_size_max** (p. 1789) across *all* of the transports being used that are not doing zero-copy.

By default (rti::core::LENGTH_AUTO): the size is equal to the maximum **NDDS_Transport_Property_t::message_↩ size_max** (p. 1789) across *all* of the non-zero-copy transports.

You may want the value to be greater than the default if you try to limit the largest data packet that can be sent through the transport(s) in one application, but you still want to receive data from other applications that have not made the same change.

For example, to avoid IP fragmentation, you may want to set **NDDS_Transport_Property_t::message_size_max** (p. 1789) for IP-based transports to a small value, such as 1400 bytes. However, you may not be able to apply this change to all the applications at the same time. To receive data from these other applications the buffer_size should be equal to the original **NDDS_Transport_Property_t::message_size_max** (p. 1789).

[default] rti::core::LENGTH_AUTO

[range] [1, 1 GB] or rti::core::LENGTH_AUTO

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.296.4.5 buffer_size() [2/2]

```
int32_t rti::core::policy::ReceiverPool::buffer_size ( ) const
```

Getter (see setter with the same name)

8.296.4.6 buffer_alignment() [1/2]

```
ReceiverPool & rti::core::policy::ReceiverPool::buffer_alignment (
            int32_t the_buffer_alignment )
```

[DEPRECATED] (p. **296)** Sets the receive buffer alignment

The receive buffer is used by the receive thread to store the raw data that arrives over the transports in non-zero-copy transports.

Zero-copy transports do not copy their data into the buffer provided by the receive thread. Instead, they provide the receive thread data in buffers allocated by the transports themselves. Only the shared memory built-in transport (SHMEM) supports zero-copy.

buffer_size must always be at least as large as the maximum **NDDS_Transport_Property_t::message_size_max** (p. 1789) across *all* of the transports being used that are not doing zero-copy.

By default (rti::core::LENGTH_AUTO): the size is equal to the maximum **NDDS_Transport_Property_t::message_↩ size_max** (p. 1789) across *all* of the non-zero-copy transports.

You may want the value to be greater than the default if you try to limit the largest data packet that can be sent through the transport(s) in one application, but you still want to receive data from other applications that have not made the same change.

For example, to avoid IP fragmentation, you may want to set **NDDS_Transport_Property_t::message_size_max** (p. 1789) for IP-based transports to a small value, such as 1400 bytes. However, you may not be able to apply this change to all the applications at the same time. To receive data from these other applications the buffer_size should be equal to the original **NDDS_Transport_Property_t::message_size_max** (p. 1789).

[default] rti::core::LENGTH_AUTO

[range] [1, 1 GB] or rti::core::LENGTH_AUTO

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.296.4.7 buffer_alignment() [2/2]

```
int32_t rti::core::policy::ReceiverPool::buffer_alignment ( ) const
```

Getter (see setter with the same name)

8.297 dds::core::Reference< DELEGATE > Class Template Reference

Base class for all reference types.

`#include <Reference.hpp>`

8.297.1 Detailed Description

template<typename DELEGATE>
class dds::core::Reference< DELEGATE >

Base class for all reference types.

The behavior and members of this class are described here: <<***reference-type***>> *(*p. *351)*

See also

<<***reference-type***>> *(*p. *351)*

8.298 dds::core::policy::Reliability Class Reference

Indicates the level of reliability in sample delivered that a **dds::pub::DataWriter** (p. 1138) offers or a **dds::sub::Data↩ Reader** (p. 976) requests.

`#include <dds/core/policy/CorePolicy.hpp>`

Public Member Functions

- **Reliability** ()

  Creates a best-effort policy.
- **Reliability** (**dds::core::policy::ReliabilityKind** the_kind, const **dds::core::Duration** &the_max_blocking_↩ time= **dds::core::Duration::from_millisecs**(100))

  Creates an instance with the specified reliability kind an optionally a specific maximum blocking time.
- **Reliability** & **kind** (**dds::core::policy::ReliabilityKind** the_kind)

  Sets the reliability kind.
- **dds::core::policy::ReliabilityKind kind** () const

  Getter (see setter with the same name)
- **Reliability** & **max_blocking_time** (const **dds::core::Duration** &d)

  Sets the maximum time a DataWriter may block on a call to ***write()*** *(*p. *1179).*
- **dds::core::Duration max_blocking_time** () const

  Getter (see setter with the same name)
- **dds::core::policy::Reliability** & **acknowledgment_kind** (**rti::core::policy::AcknowledgmentKind** ack_kind)

  <<***extension***>> *(*p. *116) Sets the kind of reliable acknowledgment*
- **rti::core::policy::AcknowledgmentKind acknowledgment_kind** () const

<<***extension***>> (p. *116) Getter (see setter with the same name)*

- **dds::core::policy::Reliability** & **instance_state_consistency_kind** (**rti::core::policy::InstanceState↩ ConsistencyKind** isr_kind)

  <<***extension***>> (p. *116) Whether instance state consistency is enabled*
- **rti::core::policy::InstanceStateConsistencyKind instance_state_consistency_kind** () const

  <<***extension***>> (p. *116) Getter (see setter with the same name)*
- **dds::core::policy::Reliability** & **exact_reliability_kind_match** (bool exact_reliability_kind_match)

  <<***extension***>> (p. *116) Sets whether or not the reliability kind should match exactly when checking compatibility between DataReaders and DataWriters.*
- bool **exact_reliability_kind_match** () const

  <<***extension***>> (p. *116) Getter (see setter with the same name)*

Static Public Member Functions

- static **Reliability Reliable** (const **dds::core::Duration** & **max_blocking_time**= **dds::core::Duration::from↩ _millisecs**(100))

  Creates a policy with ReliabilityKind::RELIABLE and optionally a max blocking time.
- static **Reliability BestEffort** ()

  Creates a policy with ReliabilityKind::BEST_EFFORT.

8.298.1 Detailed Description

Indicates the level of reliability in sample delivered that a **dds::pub::DataWriter** (p. 1138) offers or a **dds::sub::Data↩ Reader** (p. 976) requests.

Entity:

dds::topic::Topic (p. 2474), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Status:

dds::core::status::StatusMask::offered_incompatible_qos() (p. 2380), **dds::core::status::StatusMask↩ ::requested_incompatible_qos()** (p. 2381)

Properties:

RxO (p. 514) = YES
Changeable (p. 514) = **UNTIL ENABLE** (p. 514) (the instance_state_consistency_kind is **Changeable** (p. 514) = **NO** (p. 514))

8.298.2 Usage

This policy indicates the level of reliability requested by a **dds::sub::DataReader** (p. 976) or offered by a **dds::pub::DataWriter** (p. 1138).

The reliability of a connection between a DataWriter and DataReader is entirely user configurable. It can be done on a per DataWriter/DataReader connection. A connection may be configured to be "best effort" which means that RTI Connext will not use any resources to monitor or guarantee that the data sent by a DataWriter is received by a DataReader.

For some use cases, such as the periodic update of sensor values to a GUI displaying the value to a person, **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157) delivery is often good enough. It is certainly the fastest, most efficient, and least resource-intensive (CPU and network bandwidth) method of getting the newest/latest value for a topic from DataWriters to DataReaders. But there is no guarantee that the data sent will be received. It may be lost due to a variety of factors, including data loss by the physical transport such as wireless RF or even Ethernet.

However, there are data streams (topics) in which you want an absolute guarantee that all data sent by a DataWriter is received reliably by DataReaders. This means that RTI Connext must check whether or not data was received, and repair any data that was rejected by resending a copy of the data as many times as it takes for the DataReader to receive the data. RTI Connext uses a reliability protocol configured and tuned by these QoS policies: **dds::core::policy::History** (p. 1602), **rti::core::policy::DataWriterProtocol** (p. 1210), **rti::core::policy::DataReaderProtocol** (p. 1059), and **dds::core::policy::ResourceLimits** (p. 2203).

The **Reliability** (p. 2149) QoS policy is simply a switch to turn on the reliability protocol for a DataWriter/DataReader connection. The level of reliability provided by RTI Connext is determined by the configuration of the aforementioned QoS policies.

You can configure RTI Connext to deliver *all* data in the order they were sent (also known as absolute or strict reliability). Or, as a tradeoff for less memory, CPU, and network usage, you can choose a reduced level of reliability where only the last *N* values are guaranteed to be delivered reliably to DataReaders (where *N* is user-configurable). In the reduced level of reliability, there are no guarantees that the data sent before the last *N* are received. Only the last *N* data packets are monitored and repaired if necessary.

These levels are ordered, **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157) $<$ **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157). A **dds::pub::DataWriter** (p. 1138) offering one level is implicitly offering all levels below.

The setting of this policy has a dependency on the setting of the **RESOURCE_LIMITS** (p. 546) policy. In case the reliability kind is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) the write operation on the **dds::pub::DataWriter** (p. 1138) may block if the modification would cause data to be lost or else cause one of the limits in specified in the **RESOURCE_LIMITS** (p. 546) to be exceeded. Under these circumstances, the **RELIABILITY** (p. 544) `max_blocking_time` configures the maximum duration the write operation may block.

If the **dds::core::policy::Reliability::kind** (p. 2153) is set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157), data samples originating from a single **dds::pub::DataWriter** (p. 1138) cannot be made available to the **dds::sub::DataReader** (p. 976) if there are previous data samples that have not been received yet due to a communication error. In other words, RTI Connext will repair the error and resend data samples as needed in order to reconstruct a correct snapshot of the **dds::pub::DataWriter** (p. 1138) history before it is accessible by the **dds::sub::DataReader** (p. 976).

If the **dds::core::policy::Reliability::kind** (p. 2153) is set to **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157), the service will not re-transmit missing data samples. However, for data samples originating from any one DataWriter the service will ensure they are stored in the **dds::sub::DataReader** (p. 976) history in the same order they originated in the **dds::pub::DataWriter** (p. 1138). In other words, the **dds::sub::DataReader** (p. 976) may miss some data samples, but it will never see the value of a data object change from a newer value to an older value.

See also

dds::core::policy::History (p. 1602)

dds::core::policy::ResourceLimits (p. 2203)

8.298.3 Compatibility

The value offered is considered compatible with the value requested if and only if:

- The inequality *offered* `kind` >= *requested* `kind` evaluates to 'TRUE'. For the purposes of this inequality, the values of **dds::core::policy::Reliability::kind** (p. 2153) are considered ordered such that **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157) < **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157). If **dds::core::policy::Reliability::exact_reliability_kind_match** (p. 2155) is set to true, then only exact kind matches, *offered* `kind` == *requested* `kind`, are considered compatible.

- The offered acknowledgment_kind = **rti::core::policy::AcknowledgmentKind_def::PROTOCOL** (p. 797) and requested acknowledgment_kind = **rti::core::policy::AcknowledgmentKind_def::PROTOCOL** (p. 797) OR offered acknowledgment_kind != **rti::core::policy::AcknowledgmentKind_def::PROTOCOL** (p. 797).

- The inequality *offered* `instance_state_consistency_kind` >= *requested* `instance_state_consistency_kind` evaluates to 'TRUE'. For the purposes of this inequaility, the values of **dds::core::policy::Reliability::instance_state_consistency_kind** (p. 2155) are considered ordered such that **rti::core::policy::InstanceStateConsistencyKind::none** (p. 545) < **rti::core::policy::InstanceStateConsistencyKind::recover_state** (p. 545).

8.298.4 Constructor & Destructor Documentation

8.298.4.1 Reliability() [1/2]

```
dds::core::policy::Reliability::Reliability ( )  [inline]
```

Creates a best-effort policy.

8.298.4.2 Reliability() [2/2]

```
dds::core::policy::Reliability::Reliability (
             dds::core::policy::ReliabilityKind the_kind,
            const  dds::core::Duration & the_max_blocking_time =  dds::core::Duration::from_millisecs(100) )  [inline]
```

Creates an instance with the specified reliability kind an optionally a specific maximum blocking time.

The max blocking time only applies to reliable DataWriters.

8.298.5 Member Function Documentation

8.298.5.1 kind() [1/2]

```
Reliability & dds::core::policy::Reliability::kind (
            dds::core::policy::ReliabilityKind the_kind )  [inline]
```

Sets the reliability kind.

[default] **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157) for **dds::sub::DataReader** (p. 976) and **dds::topic::Topic** (p. 2474), **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157) for **dds::pub::DataWriter** (p. 1138)

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Reliability** (p. 222)

8.298.5.2 kind() [2/2]

```
dds::core::policy::ReliabilityKind dds::core::policy::Reliability::kind ( ) const  [inline]
```

Getter (see setter with the same name)

8.298.5.3 max_blocking_time() [1/2]

```
Reliability & dds::core::policy::Reliability::max_blocking_time (
           const  dds::core::Duration & d )  [inline]
```

Sets the maximum time a DataWriter may block on a call to **write()** (p. 1179).

See Writing Data, in the Core Libraries User's Manual, for more information about when the **write()** (p. 1179) call can block and what happens when it times out.

Has no meaning for DataReaders.

[default] 100 milliseconds

[range] [0,1 year] or **dds::core::Duration::infinite()** (p. 1451)

See also

dds::core::policy::ResourceLimits (p. 2203)

Categories

Immutable (p. 243), **Discovery Metadata** (p. 288), **Reliability** (p. 222), **Flow Control** (p. 188)

8.298.5.4 max_blocking_time() [2/2]

```
dds::core::Duration dds::core::policy::Reliability::max_blocking_time ( ) const  [inline]
```

Getter (see setter with the same name)

8.298.5.5 Reliable()

```
static  Reliability dds::core::policy::Reliability::Reliable (
            const  dds::core::Duration & max_blocking_time =  dds::core::Duration::from_millisecs(100)
)  [inline], [static]
```

Creates a policy with ReliabilityKind::RELIABLE and optionally a max blocking time.

8.298.5.6 BestEffort()

```
static  Reliability dds::core::policy::Reliability::BestEffort ( )  [inline], [static]
```

Creates a policy with ReliabilityKind::BEST_EFFORT.

8.298.5.7 acknowledgment_kind() [1/2]

```
dds::core::policy::Reliability & acknowledgment_kind (
            rti::core::policy::AcknowledgmentKind ack_kind )
```

<<***extension***>> *(*p. *116)* Sets the kind of reliable acknowledgment

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This setting applies only to the case where **dds::core::policy::Reliability::kind** (p. 2153) = **dds::core::policy::↩ ReliabilityKind_def::RELIABLE** (p. 2157).

Sets the kind acknowledgments supported by a **dds::pub::DataWriter** (p. 1138) and sent by **dds::sub::DataReader** (p. 976).

[default] rti::core::policy::AcknowledgmentKind_def::PROTOCOL (p. 797)

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Reliability** (p. 222), **Application Acknowledgement** (p. 158)

8.298.5.8 acknowledgment_kind() [2/2]

```
rti::core::policy::AcknowledgmentKind acknowledgment_kind ( ) const
```

<<***extension***>> *(*p. *116)* Getter (see setter with the same name)

8.298.5.9 instance_state_consistency_kind() [1/2]

```
dds::core::policy::Reliability & instance_state_consistency_kind (
            rti::core::policy::InstanceStateConsistencyKind isr_kind )
```

<<***extension***>> *(*p. *116)* Whether instance state consistency is enabled

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

A DataReader that determines that the DataWriter is no longer alive will transition instances to NOT_ALIVE_NO_↩ WRITERS if there are no other DataWriters updating that instance. If the DataReader rediscovers the DataWriter, the DataReader does not automatically transition instances back to the state they were in prior to the disconnection until it gets updates (i.e., samples) for them.

If `InstanceStateConsistencyKind` is set to `RECOVER_INSTANCE_STATE_CONSISTENCY`, then when the DataReader rediscovers a DataWriter, the DataReader will query the DataWriter for the state of its instances, and restore the instances to their correct state.

[default] rti::core::policy::InstanceStateConsistencyKind::none (p. 545) for **dds::sub::DataReader** (p. 976), **rti↩ ::core::policy::InstanceStateConsistencyKind::recover_state** (p. 545) for **dds::pub::DataWriter** (p. 1138)

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Reliability** (p. 222), **Instance Lifecycle** (p. 193)

8.298.5.10 instance_state_consistency_kind() [2/2]

```
rti::core::policy::InstanceStateConsistencyKind instance_state_consistency_kind ( ) const
```

<<***extension***>> *(*p. *116)* Getter (see setter with the same name)

8.298.5.11 exact_reliability_kind_match() [1/2]

```
dds::core::policy::Reliability & exact_reliability_kind_match (
            bool exact_reliability_kind_match )
```

<<***extension***>> *(*p. *116)* Sets whether or not the reliability kind should match exactly when checking compatibility between DataReaders and DataWriters.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

If set to TRUE in the DataWriter, the offered and requested reliability kinds must be exactly the same to be considered compatible for endpoint matching. If set to FALSE in the DataWriter, the offered reliability kind must be greater than or equal to the requested reliability kind to be considered compatible for endpoint matching. Set it to TRUE in the DataWriter to save memory and CPU cycles if communication with **dds::core::policy::ReliabilityKind_def::BEST_↩ EFFORT** (p. 2157) DataReaders is not required.

This setting has no effect on DataReaders.

[default] FALSE

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Reliability** (p. 222)

8.298.5.12 exact_reliability_kind_match() [2/2]

```
bool exact_reliability_kind_match ( ) const
```

<<***extension***>> *(*p. *116)* Getter (see setter with the same name)

8.299 dds::core::policy::ReliabilityKind_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) ReliabilityKind.

```
#include <PolicyKind.hpp>
```

Public Types

- enum **type** {
  BEST_EFFORT ,
  RELIABLE }

  The underlying `enum` *type.*

8.299.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) ReliabilityKind.

8.299.2 Member Enumeration Documentation

8.299.2.1 type

`enum` **dds::core::policy::ReliabilityKind_def::type**

The underlying `enum` type.

Enumerator

BEST_EFFORT	Indicates that it is acceptable to not retry propagation of any samples. Presumably new values for the samples are generated often enough that it is not necessary to re-send or acknowledge any samples. **[default]** for **dds::sub::DataReader** (p. 976) and **dds::topic::Topic** (p. 2474)
RELIABLE	Specifies that RTI Connext will attempt to deliver all samples in its history. Missed samples may be retried. In steady-state (no modifications communicated via the **dds::pub::DataWriter** (p. 1138)), RTI Connext guarantees that all samples in the **dds::pub::DataWriter** (p. 1138) history will eventually be delivered to all the **dds::sub::DataReader** (p. 976) objects (subject to timeouts that indicate loss of communication with a particular **dds::sub::Subscriber** (p. 2410)). Outside steady state, the **HISTORY** (p. 533) and **RESOURCE_LIMITS** (p. 546) policies will determine how samples become part of the history and whether samples can be discarded from it. **[default]** for **dds::pub::DataWriter** (p. 1138)

8.300 rti::core::status::ReliableReaderActivityChangedStatus Class Reference

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::reliable_reader_activity_changed()** (p. 2385)

`#include <Status.hpp>`

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **EventCount32 active_count** () const

  The current number of reliable readers currently matched with this reliable writer.
- **EventCount32 inactive_count** () const

  The number of reliable readers that have been dropped by this reliable writer because they failed to send acknowledgements in a timely fashion.
- **dds::core::InstanceHandle last_instance_handle** () const

  The instance handle of the last reliable remote reader to be determined inactive.

8.300.1 Detailed Description

<<***extension***>> *(*p. *354)* Information about the status **dds::core::status::StatusMask::reliable_reader_activity_changed()** (p. 2385)

Entity:

dds::pub::DataWriter (p. 1138)

Listener:

dds::pub::DataWriterListener (p. 1202)

This status is the reciprocal status to the **dds::core::status::LivelinessChangedStatus** (p. 1652) on the reader. It is different than the **dds::core::status::LivelinessLostStatus** (p. 1655) on the writer in that the latter informs the writer about its own liveliness; this status informs the writer about the "liveliness" (activity) of its matched readers.

All counts in this status will remain at zero for best effort writers.

8.300.2 Member Function Documentation

8.300.2.1 active_count()

```
EventCount32 rti::core::status::ReliableReaderActivityChangedStatus::active_count ( ) const  [inline]
```

The current number of reliable readers currently matched with this reliable writer.

8.300.2.2 inactive_count()

```
EventCount32 rti::core::status::ReliableReaderActivityChangedStatus::inactive_count ( ) const
[inline]
```

The number of reliable readers that have been dropped by this reliable writer because they failed to send acknowledgements in a timely fashion.

A reader is considered to be inactive after is has been sent heartbeats **rti::core::policy::RtpsReliableWriterProtocol↩ ::max_heartbeat_retries** (p. 2239) times, each heartbeat having been separated from the previous by the current heartbeat period.

8.300.2.3 last_instance_handle()

```
dds::core::InstanceHandle rti::core::status::ReliableReaderActivityChangedStatus::last_instance↩
_handle ( ) const  [inline]
```

The instance handle of the last reliable remote reader to be determined inactive.

8.301 rti::core::status::ReliableWriterCacheChangedStatus Class Reference

<<***extension***>> *(*p. *354)* Information about the status **dds::core::status::StatusMask::reliable_writer_cache_↩ changed()** (p. 2385)

```
#include <Status.hpp>
```

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **EventCount32 empty_reliable_writer_cache** () const

  The number of times the reliable writer's cache of unacknowledged samples has become empty.
- **EventCount32 full_reliable_writer_cache** () const

  The number of times the reliable writer's cache, or send window, of unacknowledged samples has become full.
- **EventCount32 low_watermark_reliable_writer_cache** () const

  The number of times the reliable writer's cache of unacknowledged samples has fallen to the low watermark.
- **EventCount32 high_watermark_reliable_writer_cache** () const

  The number of times the reliable writer's cache of unacknowledged samples has risen to the high watermark.
- int32_t **unacknowledged_sample_count** () const

  The current number of unacknowledged samples in the writer's cache.
- int32_t **unacknowledged_sample_count_peak** () const

  The highest value that unacknowledged_sample_count has reached until now.
- int64_t **replaced_unacknowledged_sample_count** () const

  The number of unacknowledged samples that have been replaced in the writer's cache.

8.301.1 Detailed Description

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385)

Entity:

dds::pub::DataWriter (p. 1138)

Listener:

dds::pub::DataWriterListener (p. 1202)

A written sample is unacknowledged (and therefore accounted for in this status) if the writer is reliable and one or more readers matched with the writer has not yet sent an acknowledgement to the writer declaring that it has received the sample.

If the low watermark is zero and the unacknowledged sample count decreases to zero, both the low watermark and cache empty events are considered to have taken place. A single callback will be dispatched (assuming the user has requested one) that contains both status changes. The same logic applies when the high watermark is set equal to the maximum number of samples and the cache becomes full.

8.301.2 Member Function Documentation

8.301.2.1 empty_reliable_writer_cache()

```
EventCount32 rti::core::status::ReliableWriterCacheChangedStatus::empty_reliable_writer_cache ( )
const  [inline]
```

The number of times the reliable writer's cache of unacknowledged samples has become empty.

8.301.2.2 full_reliable_writer_cache()

```
EventCount32 rti::core::status::ReliableWriterCacheChangedStatus::full_reliable_writer_cache ( )
const  [inline]
```

The number of times the reliable writer's cache, or send window, of unacknowledged samples has become full.

Applies to writer's cache when the send window is enabled (when both **rti::core::policy::RtpsReliableWriterProtocol::min_send_window_size** (p. 2249) and **rti::core::policy::RtpsReliableWriterProtocol::max_send_window_size** (p. 2250) are not **dds::core::LENGTH_UNLIMITED** (p. 438)).

Otherwise, applies when the number of unacknowledged samples has reached the send window limit.

8.301.2.3 low_watermark_reliable_writer_cache()

```
EventCount32 rti::core::status::ReliableWriterCacheChangedStatus::low_watermark_reliable_writer↩
_cache ( ) const  [inline]
```

The number of times the reliable writer's cache of unacknowledged samples has fallen to the low watermark.

A low watermark event will only be considered to have taken place when the number of unacknowledged samples in the writer's cache *decreases* to this value. A sample count that increases to this value will not result in a callback or in a change to the total count of low watermark events.

When the writer's send window is enabled, the low watermark is scaled down, if necessary, to fit within the current send window.

Set the low watermark value using the **rti::core::policy::RtpsReliableWriterProtocol::low_watermark** (p. 2232) field of the **rti::core::policy::RtpsReliableWriterProtocol** (p. 2229) structure.

8.301.2.4 high_watermark_reliable_writer_cache()

```
EventCount32 rti::core::status::ReliableWriterCacheChangedStatus::high_watermark_reliable_↩
writer_cache ( ) const  [inline]
```

The number of times the reliable writer's cache of unacknowledged samples has risen to the high watermark.

A high watermark event will only be considered to have taken place when the number of unacknowledged sampled *increases* to this value. A sample count that was above this value and then decreases back to it will not trigger an event.

When the writer's send window is enabled, the high watermark is scaled down, if necessary, to fit within the current send window.

Set the high watermark value using the **rti::core::policy::RtpsReliableWriterProtocol::high_watermark** (p. 2233) field of the **rti::core::policy::RtpsReliableWriterProtocol** (p. 2229) structure.

8.301.2.5 unacknowledged_sample_count()

```
int32_t rti::core::status::ReliableWriterCacheChangedStatus::unacknowledged_sample_count ( ) const
[inline]
```

The current number of unacknowledged samples in the writer's cache.

A sample is considered unacknowledged if the writer has failed to receive an acknowledgement from one or more reliable readers matched to it.

8.301.2.6 unacknowledged_sample_count_peak()

```
int32_t rti::core::status::ReliableWriterCacheChangedStatus::unacknowledged_sample_count_peak ( )
const  [inline]
```

The highest value that unacknowledged_sample_count has reached until now.

8.301.2.7 replaced_unacknowledged_sample_count()

```
int64_t rti::core::status::ReliableWriterCacheChangedStatus::replaced_unacknowledged_sample_count
( ) const  [inline]
```

The number of unacknowledged samples that have been replaced in the writer's cache.

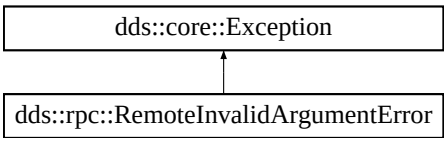
Total number of unacknowledged samples that have been replaced by a DataWriter after applying **dds::core::policy↩ ::HistoryKind_def::KEEP_LAST** (p. 1607) policy.

8.302 dds::rpc::RemoteInvalidArgumentError Class Reference

Thrown when a Client provides an invalid argument to a Service operation.

```
#include <Exception.hpp>
```

Inheritance diagram for dds::rpc::RemoteInvalidArgumentError:



Public Member Functions

- virtual const char * **what** () const noexcept

  *Access the message contained in this **RemoteInvalidArgumentError** (p. 2162) exception.*

8.302.1 Detailed Description

Thrown when a Client provides an invalid argument to a Service operation.

Inherits also from `std::logic_error`

8.302.2 Member Function Documentation

8.302.2.1 what()

```
virtual const char * dds::rpc::RemoteInvalidArgumentError::what ( ) const  [inline], [virtual],
[noexcept]
```

Access the message contained in this **RemoteInvalidArgumentError** (p. 2162) exception.

Returns

The message.

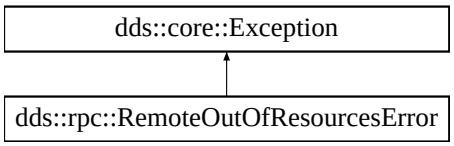
Implements **dds::core::Exception** (p. 1547).

8.303 dds::rpc::RemoteOutOfResourcesError Class Reference

Thrown when the Service runs out of resources to fulfill a Client's operation.

```
#include <Exception.hpp>
```

Inheritance diagram for dds::rpc::RemoteOutOfResourcesError:



Public Member Functions

- virtual const char * **what** () const noexcept

  *Access the message contained in this **RemoteOutOfResourcesError** (p. 2163) exception.*

8.303.1 Detailed Description

Thrown when the Service runs out of resources to fulfill a Client's operation.

Inherits also from `std::logic_error`

8.303.2 Member Function Documentation

8.303.2.1 what()

```
virtual const char * dds::rpc::RemoteOutOfResourcesError::what ( ) const  [inline], [virtual],
[noexcept]
```

Access the message contained in this **RemoteOutOfResourcesError** (p. 2163) exception.

Returns

The message.

Implements **dds::core::Exception** (p. 1547).

8.304 rti::core::policy::RemoteParticipantPurgeKind_def Struct Reference

<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) RemoteParticipantPurgeKind

`#include <PolicyKind.hpp>`

Public Types

- enum **type** {
  LIVELINESS_BASED ,
  NO_PURGE }
  The underlying `enum` *type.*

8.304.1 Detailed Description

<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) RemoteParticipantPurgeKind

Available behaviors for halting communication with remote participants (and their contained entities) with which discovery communication has been lost.

When discovery communication with a remote participant has been lost, the local participant must make a decision about whether to continue attempting to communicate with that participant and its contained entities. This "kind" is used to select the desired behavior.

This "kind" does not pertain to the situation in which a remote participant has been gracefully deleted and notification of that deletion have been successfully received by its peers. In that case, the local participant will immediately stop attempting to communicate with those entities and will remove the associated remote entity records from its internal database.

See also

rti::core::policy::DiscoveryConfig::remote_participant_purge_kind (p. 1281)

8.304.2 Member Enumeration Documentation

8.304.2.1 type

`enum` **rti::core::policy::RemoteParticipantPurgeKind_def::type**

The underlying `enum` type.

Enumerator

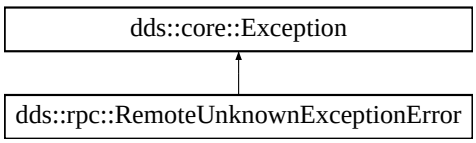
LIVELINESS_BASED	**[default]** Maintain knowledge of the remote participant for as long as it maintains its liveliness contract. A participant will continue attempting communication with its peers, even if discovery communication with them is lost, as long as the remote participants maintain their liveliness. If both discovery communication and participant liveliness are lost, however, the local participant will remove all records of the remote participant and its contained endpoints, and no further data communication with them will occur until and unless they are rediscovered. The liveliness contract a participant promises to its peers – its "liveliness lease duration" – is specified in its **rti::core::policy::DiscoveryConfig::participant_liveliness_lease_duration** (p. 1279) QoS field. It maintains that contract by writing data to those other participants with a writer that has a **dds::core::policy::LivelinessKind_def** (p. 1654) of **dds::core::policy::LivelinessKind_def::AUTOMATIC** (p. 1655) or **dds::core::policy::LivelinessKind_def::MANUAL_BY_PARTICIPANT** (p. 1655) and by asserting itself (at the **rti::core::policy::DiscoveryConfig::participant_liveliness_assert_period** (p. 1279)) over the Simple **Discovery** (p. 1265) Protocol.
NO_PURGE	Never "forget" a remote participant with which discovery communication has been lost. If a participant with this behavior loses discovery communication with a remote participant, it will nevertheless remember that remote participant and its endpoints and continue attempting to communicate with them indefinitely. This value has consequences for a participant's resource usage. If discovery communication with a remote participant is lost, but the same participant is later rediscovered, any relevant records that remain in the database will be reused. However, if it is not rediscovered, the records will continue to take up space in the database for as long as the local participant remains in existence.

8.305 dds::rpc::RemoteUnknownExceptionError Class Reference

Thrown when an unknown exception occurs in the Service.

`#include <Exception.hpp>`

Inheritance diagram for dds::rpc::RemoteUnknownExceptionError:



Public Member Functions

- virtual const char * **what** () const noexcept

  *Access the message contained in this **RemoteUnknownExceptionError** (p. 2165) exception.*

8.305.1 Detailed Description

Thrown when an unknown exception occurs in the Service.

Inherits also from `std::logic_error`

8.305.2 Member Function Documentation

8.305.2.1 what()

```
virtual const char * dds::rpc::RemoteUnknownExceptionError::what ( ) const  [inline], [virtual],
[noexcept]
```

Access the message contained in this **RemoteUnknownExceptionError** (p. 2165) exception.

Returns

The message.

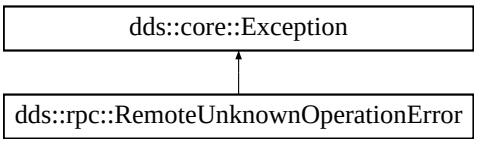
Implements **dds::core::Exception** (p. 1547).

8.306 dds::rpc::RemoteUnknownOperationError Class Reference

Thrown when a Client calls an operation that doesn't exist in the Service.

`#include <Exception.hpp>`

Inheritance diagram for dds::rpc::RemoteUnknownOperationError:



Public Member Functions

- virtual const char * **what** () const noexcept

  *Access the message contained in this **RemoteUnknownOperationError** (p. 2166) exception.*

8.306.1 Detailed Description

Thrown when a Client calls an operation that doesn't exist in the Service.

Inherits also from `std::logic_error`

8.306.2 Member Function Documentation

8.306.2.1 what()

```
virtual const char * dds::rpc::RemoteUnknownOperationError::what ( ) const  [inline], [virtual],
[noexcept]
```

Access the message contained in this **RemoteUnknownOperationError** (p. 2166) exception.

Returns

The message.

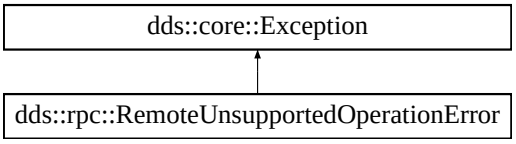
Implements **dds::core::Exception** (p. 1547).

8.307 dds::rpc::RemoteUnsupportedOperationError Class Reference

Thrown when a Client calls an operation that is not supported by the Service.

```
#include <Exception.hpp>
```

Inheritance diagram for dds::rpc::RemoteUnsupportedOperationError:



Public Member Functions

- virtual const char * **what** () const noexcept

  *Access the message contained in this **RemoteUnsupportedOperationError** (p. 2167) exception.*

8.307.1 Detailed Description

Thrown when a Client calls an operation that is not supported by the Service.

Inherits also from `std::logic_error`

8.307.2 Member Function Documentation

8.307.2.1 what()

```
virtual const char * dds::rpc::RemoteUnsupportedOperationError::what ( ) const  [inline], [virtual],
[noexcept]
```

Access the message contained in this **RemoteUnsupportedOperationError** (p. 2167) exception.

Returns

The message.

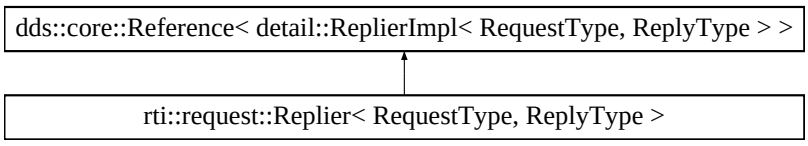
Implements **dds::core::Exception** (p. 1547).

8.308 rti::request::Replier< RequestType, ReplyType > Class Template Reference

<<***reference-type***>> *(p. 351)* Allows receiving requests and sending replies

```
#include <rti/request/Replier.hpp>
```

Inheritance diagram for rti::request::Replier< RequestType, ReplyType >:



Public Member Functions

- **Replier** (**dds::domain::DomainParticipant** participant, const std::string &service_name)
  *Creates a **Replier** (p. 2168) with the minimum set of parameters.*
- **Replier** (const **ReplierParams** &params)
  *Creates a **Replier** (p. 2168) with parameters.*
- **Replier** (const **ReplierParams** &params, **Listener** *the_listener)
  Creates a replier with additional parameters and a listener.
- **Replier** (const **ReplierParams** &params, std::shared_ptr< **Listener** > the_listener)
  Creates a replier with additional parameters and a listener.
- void **send_reply** (const ReplyType &reply, const **rti::core::SampleIdentity** &related_request_id)
  Sends a reply for a previous request.
- void **send_reply** (const ReplyType &reply, const **dds::sub::SampleInfo** &related_request_info)
  Sends a reply.
- void **send_reply** (const ReplyType &reply, **rti::pub::WriteParams** &params)
  Sends a reply with advanced parameters.
- bool **wait_for_requests** (const **dds::core::Duration** &max_wait)
  Waits for requests.
- bool **wait_for_requests** (int min_count, const **dds::core::Duration** &max_wait)
  Waits for requests.
- **dds::sub::LoanedSamples**< RequestType > **receive_requests** (const **dds::core::Duration** &max_wait)
  Waits for multiple requests and provides a loaned container to access them.
- **dds::sub::LoanedSamples**< RequestType > **receive_requests** (int min_count, const **dds::core::Duration** &max_wait)
  Waits for multiple requests and provides a loaned container to access them.
- **dds::sub::LoanedSamples**< RequestType > **take_requests** ()
  Takes all the requests.
- **dds::sub::LoanedSamples**< RequestType > **read_requests** ()
  Reads all the requests.
- **Listener** * **listener** () const
  Gets the replier listener.
- void **listener** (**Listener** *the_listener, const **dds::core::status::StatusMask** &= **dds::core::status::StatusMask::none**())
  Sets a listener to be notified when requests are available.
- std::shared_ptr< **Listener** > **get_listener** () const
  Gets the replier listener.
- void **set_listener** (std::shared_ptr< **Listener** > the_listener)
  Sets a listener to be notified when requests are available.
- **dds::sub::DataReader**< RequestType > **request_datareader** () const
  *Retrieves the underlying **dds::sub::DataReader** (p. 976).*
- **dds::pub::DataWriter**< ReplyType > **reply_datawriter** () const
  *Retrieves the underlying **dds::pub::DataWriter** (p. 1138).*

Related Functions

(Note that these are not member functions.)

- template<typename RequestType , typename ReplyType >
  size_t **matched_requester_count** (**Replier**< RequestType, ReplyType > replier)
  *Obtains the number of Requesters that match with a **Replier** (p. 2168).*

8.308.1 Detailed Description

template<typename RequestType, typename ReplyType>
class rti::request::Replier< RequestType, ReplyType >

*<<**reference-type**>>* *(p. 351)* Allows receiving requests and sending replies

Note

A **Replier** (p. 2168) provides all the functions of a *<<**reference-type**>>* *(p. 351)* except **close()** (p. 1024) and retain().

A **Replier** (p. 2168) is an entity with two associated **topics** (p. 47): a request topic and a reply topic. It can receive requests by subscribing to the request topic and can send replies to those requests by publishing the reply topic.

Valid types for these topics are: those generated by rtiddsgen, the **DDS built-in types** (p. 50), and **dds::core::xtypes::DynamicData** (p. 1461). See **Creating a Replier** (p. 347).

A **Replier** (p. 2168) has four main types of operations:

- Waiting for requests to be received from the middleware
- Getting those requests
- Receiving requests (a convenience operation that is a combination of waiting and getting in a single operation)
- Sending a reply for a previously received request (i.e., publishing a reply sample on the reply topic with special meta-data so that the orignal **Requester** (p. 2187) can identify it)

For multi-reply scenarios in which a **rti::request::Replier** (p. 2168) generates more than one reply for a request, the **rti::request::Replier** (p. 2168) should mark all the intermediate replies (all but the last) using the **rti::core::SampleFlag::intermediate_reply_sequence** (p. 2280) flag in **rti::pub::WriteParams::flag** (p. 2649).

Much like a **Requester** (p. 2187), a **Replier** (p. 2168) has an associated **dds::domain::DomainParticipant** (p. 1326), which can be shared with other Repliers or RTI Connext routines. All the other entities required for the request-reply interaction, including a **dds::pub::DataWriter** (p. 1138) for writing replies and a **dds::sub::DataReader** (p. 976) for reading requests, are automatically created when the **Replier** (p. 2168) is constructed.

Quality of Service for the underlying DataWriter and DataReader can be configured (see RequesterParams::qos_profile). By default, they are created with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157). The exact default configuration is described here: **Configuring Request-Reply QoS profiles** (p. 348)

There are several ways to use a **Replier** (p. 2168):

- A thread **receives** (p. 2175) requests and then dispatches them. If the computation of a reply is a simple operation, consider using a **SimpleReplier** (p. 2368) instead of a **Replier** (p. 2168).
- Polling without waiting, using **rti::request::Replier::take_requests** (p. 2176) directly.
- Using a **ReplierListener** (p. 2179) to get notified and **get** (p. 2176) the requests within the callback.

Template Parameters

RequestType	The data type for the request topic
ReplyType	The data type for the reply topic

See also

rti::request::Requester (p. 2187)

Request-Reply Examples (p. 344)

Basic Replier example (p. 348)

8.308.2 Constructor & Destructor Documentation

8.308.2.1 Replier() [1/4]

```
template<typename RequestType , typename ReplyType >
rti::request::Replier< RequestType, ReplyType >::Replier (
              dds::domain::DomainParticipant participant,
             const std::string & service_name )  [inline]
```

Creates a **Replier** (p. 2168) with the minimum set of parameters.

Parameters

participant	The DomainParticipant that this **Replier** (p. 2168) uses to join a domain.
service_name	The service name. See **ReplierParams::service_name** (p. 2181)

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.308.2.2 Replier() [2/4]

```
template<typename RequestType , typename ReplyType >
rti::request::Replier< RequestType, ReplyType >::Replier (
             const  ReplierParams & params )  [inline], [explicit]
```

Creates a **Replier** (p. 2168) with parameters.

Parameters

params	All the parameters that configure this **Replier** (p. 2168)

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

ReplierParams (p. 2180)

Creating a Replier (p. 347)

8.308.2.3 Replier() [3/4]

```
template<typename RequestType , typename ReplyType >
rti::request::Replier< RequestType, ReplyType >::Replier (
            const  ReplierParams & params,
             Listener * the_listener )  [inline]
```

Creates a replier with additional parameters and a listener.

[DEPRECATED] When using a listener, prefer the constructor that receives a `shared_ptr<`**Listener** (p. 1637)`>` instead of a regular `Listener*` pointer.

Parameters

params	All the parameters that configure this **Replier** (p. 2168)
the_listener	A **ReplierListener** (p. 2179) that notifies when new requests are available. Cannot be NULL.

Note

The listener must be reset with `this->listener(NULL)` before this object can be destroyed. Alternatively, create the **Replier** (p. 2168) and then use **rti::core::bind_listener()** (p. 708) to have it automatically reset.

8.308.2.4 Replier() [4/4]

```
template<typename RequestType , typename ReplyType >
rti::request::Replier< RequestType, ReplyType >::Replier (
            const  ReplierParams & params,
            std::shared_ptr<  Listener > the_listener )  [inline]
```

Creates a replier with additional parameters and a listener.

Parameters

params	All the parameters that configure this **Replier** (p. 2168)
the_listener	A **ReplierListener** (p. 2179) that notifies when new requests are available. Cannot be nullptr, but the parameter can be omitted.

8.308.3 Member Function Documentation

8.308.3.1 send_reply() [1/3]

```
template<typename RequestType , typename ReplyType >
void  rti::request::Replier< RequestType, ReplyType >::send_reply (
            const ReplyType & reply,
            const  rti::core::SampleIdentity & related_request_id )  [inline]
```

Sends a reply for a previous request.

The related request identity can be retrieved from an existing request sample (Sample).

Parameters

reply	The reply to be sent.
related_request_id	The identity of a previously received request

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

Sample::identity

rti::request::Replier::receive_requests (p. 2175)

rti::request::Replier::take_requests (p. 2176)

Basic Replier example (p. 348)

8.308.3.2 send_reply() [2/3]

```
template<typename RequestType , typename ReplyType >
void  rti::request::Replier< RequestType, ReplyType >::send_reply (
```

```
            const ReplyType & reply,
            const  dds::sub::SampleInfo & related_request_info )  [inline]
```

Sends a reply.

This operation obtains the related_request_id from a request SampleInfo, that is, it's equivalent to send_reply(reply, related_request_info->**original_publication_virtual_sample_identity()** (p. 2292))

References **dds::sub::SampleInfo::original_publication_virtual_sample_identity()**.

8.308.3.3 send_reply() [3/3]

```
template<typename RequestType , typename ReplyType >
void  rti::request::Replier< RequestType, ReplyType >::send_reply (
            const ReplyType & reply,
             rti::pub::WriteParams & params )  [inline]
```

Sends a reply with advanced parameters.

Parameters

reply	The reply to be sent
params	(in-out) The parameters to write the reply (see).

Precondition

params.related_sample_identity() must be set with the related request SampleIdentity.

Other parameters to customize are discussed in **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179). In particular, **rti::pub::WriteParams::flag** (p. 2649) can be set to **rti::core::SampleFlag::intermediate_reply_sequence** (p. 2280) to inform the **Requester** (p. 2187) application that this reply will be followed by additional replies to the same request. The **Requester** (p. 2187) can access the flag in **dds::sub::SampleInfo::flag()** (p. 2293) when it receives the reply.

8.308.3.4 wait_for_requests() [1/2]

```
template<typename RequestType , typename ReplyType >
bool  rti::request::Replier< RequestType, ReplyType >::wait_for_requests (
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for requests.

Equivalent to **rti::request::Replier::wait_for_requests** (p. 2174) with `min_count=1`

See also

rti::request::Replier::wait_for_requests (p. 2174)

8.308.3.5 wait_for_requests() [2/2]

```
template<typename RequestType , typename ReplyType >
bool  rti::request::Replier< RequestType, ReplyType >::wait_for_requests (
            int min_count,
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for requests.

This operation waits for min_count requests to be available. It will wait up to max_wait .

This operation is similar to **rti::request::Requester::wait_for_replies(int, const dds::core::Duration&)** (p. 2195).

Parameters

min_count	Minimum number of requests that need to be available for this operation to unblock.
max_wait	Maximum waiting time after which this operation unblocks regardless of how many requests are available.

Returns

true if at least min_count requests were available before max_wait elapsed, or false otherwise.

See also

rti::request::Replier::take_requests (p. 2176)

rti::request::Requester::wait_for_replies(int, const dds::core::Duration&) (p. 2195)

8.308.3.6 receive_requests() [1/2]

```
template<typename RequestType , typename ReplyType >
dds::sub::LoanedSamples< RequestType >  rti::request::Replier< RequestType, ReplyType >::receive↩
_requests (
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for multiple requests and provides a loaned container to access them.

Equivalent to using **rti::request::Replier::receive_requests** (p. 2175) with `min_count=1` and `max_count=`**dds↩ ::core::LENGTH_UNLIMITED** (p. 438)

See also

dds::sub::LoanedSamples (p. 1663)

rti::request::Replier::receive_requests (p. 2175)

8.308.3.7 receive_requests() [2/2]

```
template<typename RequestType , typename ReplyType >
dds::sub::LoanedSamples< RequestType >  rti::request::Replier< RequestType, ReplyType >::receive_requests (
            int min_count,
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for multiple requests and provides a loaned container to access them.

Equivalent to using **rti::request::Replier::wait_for_requests** (p. 2174) and **rti::request::Replier::take_requests** (p. 2176)

See also

dds::sub::LoanedSamples (p. 1663)

rti::request::Replier::wait_for_requests (p. 2174)

rti::request::Replier::take_requests (p. 2176)

8.308.3.8 take_requests()

```
template<typename RequestType , typename ReplyType >
dds::sub::LoanedSamples< RequestType >  rti::request::Replier< RequestType, ReplyType >::take_requests ( )  [inline]
```

Takes all the requests.

See also

rti::request::Requester::take_replies (p. 2193)

8.308.3.9 read_requests()

```
template<typename RequestType , typename ReplyType >
dds::sub::LoanedSamples< RequestType >  rti::request::Replier< RequestType, ReplyType >::read_requests ( )  [inline]
```

Reads all the requests.

This operation is equivalent to **rti::request::Replier::take_requests** (p. 2176) except the requests remain in the **Replier** (p. 2168) and can be read or taken again.

8.308.3.10 listener() [1/2]

```
template<typename RequestType , typename ReplyType >
Listener * rti::request::Replier< RequestType, ReplyType >::listener ( ) const [inline]
```

Gets the replier listener.

[DEPRECATED] Prefer **get_listener()** (p. 2177) instead of this function.

8.308.3.11 listener() [2/2]

```
template<typename RequestType , typename ReplyType >
void rti::request::Replier< RequestType, ReplyType >::listener (
            Listener * the_listener,
            const dds::core::status::StatusMask & = dds::core::status::StatusMask::none() )
[inline]
```

Sets a listener to be notified when requests are available.

[DEPRECATED] The use of **set_listener()** (p. 2177) is recommended. Unlike this function, set_listener receives a shared_ptr which simplifies the management of listener's lifecycle.

The listener will be called when new requests are available.

Note

rti::core::bind_listener() (p. 708) is recommended instead of this listener setter. **bind_listener()** (p. 1644) and **bind_and_manage_listener()** (p. 1644) automatically remove the listener, which allows the **Replier** (p. 2168) to be automatically destroyed.

(The second parameter is ignored)

8.308.3.12 get_listener()

```
template<typename RequestType , typename ReplyType >
std::shared_ptr< Listener > rti::request::Replier< RequestType, ReplyType >::get_listener ( )
const [inline]
```

Gets the replier listener.

The listener will be called when new requests are available

8.308.3.13 set_listener()

```
template<typename RequestType , typename ReplyType >
void rti::request::Replier< RequestType, ReplyType >::set_listener (
            std::shared_ptr< Listener > the_listener ) [inline]
```

Sets a listener to be notified when requests are available.

Warning

It's recommended that the listener implementation doesn't hold a permanent reference to the replier. If it does, the application needs to manually reset the listener (set_listener(nullptr)) to ensure that there is no cycle that prevents the destruction of these two objects.

Parameters

the_listener	A shared pointer to the listener to receive updates or `nullptr` to reset the current listener and stop receiving updates.

8.308.3.14 request_datareader()

```
template<typename RequestType , typename ReplyType >
dds::sub::DataReader< RequestType > rti::request::Replier< RequestType, ReplyType >::request_↩
datareader ( ) const [inline]
```

Retrieves the underlying **dds::sub::DataReader** (p. 976).

See also

rti::request::Requester::get_reply_datareader

Referenced by **rti::request::Replier< RequestType, ReplyType >::matched_requester_count()**.

8.308.3.15 reply_datawriter()

```
template<typename RequestType , typename ReplyType >
dds::pub::DataWriter< ReplyType > rti::request::Replier< RequestType, ReplyType >::reply_↩
datawriter ( ) const [inline]
```

Retrieves the underlying **dds::pub::DataWriter** (p. 1138).

See also

rti::request::Requester::get_request_datawriter

Referenced by **rti::request::Replier< RequestType, ReplyType >::matched_requester_count()**.

8.308.4 Friends And Related Function Documentation

8.308.4.1 matched_requester_count()

```
template<typename RequestType , typename ReplyType >
size_t matched_requester_count (
              Replier< RequestType, ReplyType > replier )  [related]
```

Obtains the number of Requesters that match with a **Replier** (p. 2168).

Note

This is a standalone function in the namespace rti::request

For a **Requester** (p. 2187) to be included in the total count, the **Requester** (p. 2187)'s DataReader or DataWriter role name (in **rti::core::policy::EntityName** (p. 1530)) must not have been modified.

References **rti::request::Replier< RequestType, ReplyType >::reply_datawriter()**, and **rti::request::Replier< RequestType, ReplyType >::request_datareader()**.

8.309 rti::request::ReplierListener< RequestType, ReplyType > Class Template Reference

Called when a **Replier** (p. 2168) has new available requests.

```
#include <rti/request/ReplierListener.hpp>
```

Public Member Functions

- virtual void **on_request_available** (**Replier**< RequestType, ReplyType > &replier)=0
  User callback.

8.309.1 Detailed Description

template<typename RequestType, typename ReplyType>
class rti::request::ReplierListener< RequestType, ReplyType >

Called when a **Replier** (p. 2168) has new available requests.

A replier listener is a way to implement a callback that will be invoked when requests are available.

This listener simply notifies when requests are available. The callback implementation can then use **rti::request::Replier::take_requests** (p. 2176) to retrieve them.

A simpler callback mechanism, where one request sample is a parameter and the reply is the callback return value, is implemented by the **SimpleReplier** (p. 2368).

See also

rti::request::Replier::Replier(const ReplierParams&) (p. 2171)

8.309.2 Member Function Documentation

8.309.2.1 on_request_available()

```
template<typename RequestType , typename ReplyType >
virtual void  rti::request::ReplierListener< RequestType, ReplyType >::on_request_available (
              Replier< RequestType, ReplyType > & replier )  [pure virtual]
```

User callback.

See also

dds::sub::DataReaderListener::on_data_available (p. 1058)

8.310 rti::request::ReplierParams Class Reference

Contains the parameters for creating a **rti::request::Replier** (p. 2168).

```
#include <ReplierParams.hpp>
```

Inherits rti::request::detail::EntityParamsWithSetters< ActualEntity >.

Public Member Functions

- **ReplierParams** (**dds::domain::DomainParticipant** participant)
  *Creates a **ReplierParams** (p. 2180) with the parameters that a **Replier** (p. 2168) always needs.*
- **ReplierParams** & **service_name** (const std::string &name)
  *The service name the **Replier** (p. 2168) offers and Requesters use to match.*
- **ReplierParams** & **request_topic_name** (const std::string &name)
  Sets a specific request topic name.
- **ReplierParams** & **reply_topic_name** (const std::string &name)
  Sets a specific reply topic name.
- **ReplierParams** & **datawriter_qos** (const **dds::core::optional**< **dds::pub::qos::DataWriterQos** > &qos)
  Sets the quality of service of the reply DataWriter.
- **ReplierParams** & **datareader_qos** (const **dds::core::optional**< **dds::sub::qos::DataReaderQos** > &qos)
  Sets the quality of service of the request DataReader.
- **ReplierParams** & **publisher** (**dds::pub::Publisher** publisher)
  Sets a specific Publisher.
- **ReplierParams** & **subscriber** (**dds::sub::Subscriber** subscriber)
  Sets a specific Subscriber.
- **ReplierParams** & **request_type** (const **dds::core::optional**< **dds::core::xtypes::DynamicType** > &type)
  The request type, when DynamicData is used.
- **ReplierParams** & **reply_type** (const **dds::core::optional**< **dds::core::xtypes::DynamicType** > &type)
  The reply type, when DynamicData is used.
- **ReplierParams** & **service_instance_name** (const **dds::core::optional**< std::string > &name)
  *The service instance name of the **Replier** (p. 2168) or the service instance the **Requester** (p. 2187) is bound to at creation.*

8.310.1 Detailed Description

Contains the parameters for creating a **rti::request::Replier** (p. 2168).

See also

RequesterParams (p. 2199)

8.310.2 Constructor & Destructor Documentation

8.310.2.1 ReplierParams()

```
rti::request::ReplierParams::ReplierParams (
            dds::domain::DomainParticipant participant )  [inline], [explicit]
```

Creates a **ReplierParams** (p. 2180) with the parameters that a **Replier** (p. 2168) always needs.

In addition to the participant , a **Replier** (p. 2168) needs either:

- A service name (**ReplierParams::service_name** (p. 2181)), or
- Custom topic names (**ReplierParams::reply_topic_name** (p. 2182) and **ReplierParams::request_topic_name** (p. 2182)).

When **dds::core::xtypes::DynamicData** (p. 1461) is used, **ReplierParams::request_type** (p. 2183) and/or **ReplierParams::reply_type** (p. 2183) are required as well.

The other parameters are optional.

Parameters

participant	The DomainParticipant that this replier uses to join a domain.

8.310.3 Member Function Documentation

8.310.3.1 service_name()

```
ReplierParams & rti::request::ReplierParams::service_name (
            const std::string & name )
```

The service name the **Replier** (p. 2168) offers and Requesters use to match.

See also

RequesterParams::service_name (p. 2200)

8.310.3.2 request_topic_name()

```
ReplierParams & rti::request::ReplierParams::request_topic_name (
            const std::string & name )
```

Sets a specific request topic name.

The specified topic name will be used, instead of allowing a topic name to be generated based on the **ReplierParams::service_name** (p. 2181). If that topic already exists, it will be reused. This is useful to have the **Replier** (p. 2168) use a **dds::topic::ContentFilteredTopic** (p. 954) to receive only certain requests.

8.310.3.3 reply_topic_name()

```
ReplierParams & rti::request::ReplierParams::reply_topic_name (
            const std::string & name )
```

Sets a specific reply topic name.

The specified topic name will be used, instead of allowing a topic name to be generated based on c the **ReplierParams::service_name** (p. 2181). If that topic already exists, it will be reused.

8.310.3.4 datawriter_qos()

```
ReplierParams & rti::request::ReplierParams::datawriter_qos (
            const dds::core::optional< dds::pub::qos::DataWriterQos > & qos )
```

Sets the quality of service of the reply DataWriter.

See also

dds::core::QosProvider (p. 2025) to access Qos profiles.

8.310.3.5 datareader_qos()

```
ReplierParams & rti::request::ReplierParams::datareader_qos (
            const dds::core::optional< dds::sub::qos::DataReaderQos > & qos )
```

Sets the quality of service of the request DataReader.

See also

dds::core::QosProvider (p. 2025) to access Qos profiles.

8.310.3.6 publisher()

```
ReplierParams & rti::request::ReplierParams::publisher (
            dds::pub::Publisher publisher )
```

Sets a specific Publisher.

See also

RequesterParams::publisher (p. 2201)

8.310.3.7 subscriber()

```
ReplierParams & rti::request::ReplierParams::subscriber (
            dds::sub::Subscriber subscriber )
```

Sets a specific Subscriber.

See also

RequesterParams::subscriber (p. 2202)

8.310.3.8 request_type()

```
ReplierParams & rti::request::ReplierParams::request_type (
           const  dds::core::optional<  dds::core::xtypes::DynamicType > & type )
```

The request type, when DynamicData is used.

This is required when the **Replier** (p. 2168) RequestType is **dds::core::xtypes::DynamicData** (p. 1461). Otherwise this is ignored.

8.310.3.9 reply_type()

```
ReplierParams & rti::request::ReplierParams::reply_type (
           const  dds::core::optional<  dds::core::xtypes::DynamicType > & type )
```

The reply type, when DynamicData is used.

This is required when the **Replier** (p. 2168) ReplyType is **dds::core::xtypes::DynamicData** (p. 1461). Otherwise this is ignored.

8.310.3.10 service_instance_name()

```
ReplierParams & rti::request::ReplierParams::service_instance_name (
            const  dds::core::optional< std::string > & name )
```

The service instance name of the **Replier** (p. 2168) or the service instance the **Requester** (p. 2187) is bound to at creation.

8.311 dds::core::status::RequestedDeadlineMissedStatus Class Reference

Information about the status **dds::core::status::StatusMask::requested_deadline_missed()** (p. 2380)

```
#include <TStatus.hpp>
```

Public Member Functions

- int32_t **total_count** () const

  *Total cumulative count of the missed deadlines detected for any instance read by the **dds::sub::DataReader** (p. 976).*
- int32_t **total_count_change** () const

  The incremental number of missed deadlines detected since the last time the listener was called or the status was read.
- const **dds::core::InstanceHandle last_instance_handle** () const

  *Handle to the last instance in the **dds::sub::DataReader** (p. 976) for which a requested deadline was missed.*

8.311.1 Detailed Description

Information about the status **dds::core::status::StatusMask::requested_deadline_missed()** (p. 2380)

8.311.2 Member Function Documentation

8.311.2.1 total_count()

```
int32_t dds::core::status::RequestedDeadlineMissedStatus::total_count ( ) const  [inline]
```

Total cumulative count of the missed deadlines detected for any instance read by the **dds::sub::DataReader** (p. 976).

8.311.2.2 total_count_change()

```
int32_t dds::core::status::RequestedDeadlineMissedStatus::total_count_change ( ) const  [inline]
```

The incremental number of missed deadlines detected since the last time the listener was called or the status was read.

8.311.2.3 last_instance_handle()

```
const  dds::core::InstanceHandle dds::core::status::RequestedDeadlineMissedStatus::last_instance↩
_handle ( ) const  [inline]
```

Handle to the last instance in the **dds::sub::DataReader** (p. 976) for which a requested deadline was missed.

8.312 dds::core::status::RequestedIncompatibleQosStatus Class Reference

Information about the status **dds::core::status::StatusMask::requested_incompatible_qos()** (p. 2381)

```
#include <TStatus.hpp>
```

Public Member Functions

- int32_t **total_count** () const

  *Total cumulative count of how many times the concerned **dds::sub::DataReader** (p. 976) discovered a **dds::pub::Data↩ Writer** (p. 1138) for the same **dds::topic::Topic** (p. 2474) with an offered QoS that is incompatible with that requested by the **dds::sub::DataReader** (p. 976).*
- int32_t **total_count_change** () const

  The change in `total_count` since the last time the listener was called or the status was read.
- **dds::core::policy::QosPolicyId last_policy_id** () const

  *The **dds::core::policy::QosPolicyId** (p. 514) of one of the policies that was found to be incompatible the last time an incompatibility was detected.*
- const **dds::core::policy::QosPolicyCountSeq policies** () const

  *A list containing, for each policy, the total number of times that the concerned **dds::sub::DataReader** (p. 976) discovered a **dds::pub::DataWriter** (p. 1138) for the same **dds::topic::Topic** (p. 2474) with an offered QoS that is incompatible with that requested by the **dds::sub::DataReader** (p. 976).*

8.312.1 Detailed Description

Information about the status **dds::core::status::StatusMask::requested_incompatible_qos()** (p. 2381)

See also

DURABILITY (p. 529)
PRESENTATION (p. 540)
RELIABILITY (p. 544)
OWNERSHIP (p. 538)
LIVELINESS (p. 535)
DEADLINE (p. 523)
LATENCY_BUDGET (p. 534)
DESTINATION_ORDER (p. 524)

8.312.2 Member Function Documentation

8.312.2.1 total_count()

```
int32_t dds::core::status::RequestedIncompatibleQosStatus::total_count ( ) const  [inline]
```

Total cumulative count of how many times the concerned **dds::sub::DataReader** (p. 976) discovered a **dds::pub↩ ::DataWriter** (p. 1138) for the same **dds::topic::Topic** (p. 2474) with an offered QoS that is incompatible with that requested by the **dds::sub::DataReader** (p. 976).

8.312.2.2 total_count_change()

```
int32_t dds::core::status::RequestedIncompatibleQosStatus::total_count_change ( ) const  [inline]
```

The change in `total_count` since the last time the listener was called or the status was read.

8.312.2.3 last_policy_id()

```
dds::core::policy::QosPolicyId dds::core::status::RequestedIncompatibleQosStatus::last_policy_id
( ) const  [inline]
```

The **dds::core::policy::QosPolicyId** (p. 514) of one of the policies that was found to be incompatible the last time an incompatibility was detected.

8.312.2.4 policies()

```
const dds::core::policy::QosPolicyCountSeq dds::core::status::RequestedIncompatibleQosStatus↩
::policies ( ) const [inline]
```

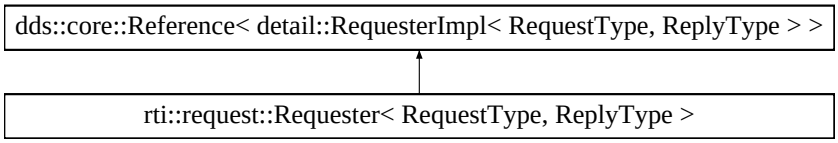
A list containing, for each policy, the total number of times that the concerned **dds::sub::DataReader** (p. 976) discovered a **dds::pub::DataWriter** (p. 1138) for the same **dds::topic::Topic** (p. 2474) with an offered QoS that is incompatible with that requested by the **dds::sub::DataReader** (p. 976).

8.313 rti::request::Requester< RequestType, ReplyType > Class Template Reference

<<***reference-type***>> *(p. 351)* Allows sending requests and receiving replies

```
#include <rti/request/Requester.hpp>
```

Inheritance diagram for rti::request::Requester< RequestType, ReplyType >:



Public Member Functions

- **Requester** (const **RequesterParams** &params)

  *Creates a **Requester** (p. 2187) with parameters.*
- **Requester** (**dds::domain::DomainParticipant** participant, const std::string &service_name)

  *Creates a **Requester** (p. 2187) with the minimum set of parameters.*
- bool **wait_for_service** (const **dds::core::Duration** &max_wait= **dds::core::Duration::infinite**())

  *Waits for one or more repliers to be discovered. This function takes advantage of enhanced discovery, making sure both the **Requester** (p. 2187) and the **Replier** (p. 2168) discover each other.*
- **rti::core::SampleIdentity** **send_request** (const RequestType &request)

  Sends a request.
- void **send_request** (const RequestType &request, **rti::pub::WriteParams** &params)

  Sends a request with advanced parameters.
- **dds::sub::LoanedSamples**< ReplyType > **receive_replies** (const **dds::core::Duration** &max_wait)

  Waits for multiple replies and provides a loaned container to access them.
- **dds::sub::LoanedSamples**< ReplyType > **receive_replies** (int min_count, const **dds::core::Duration** &max_wait)

  Waits for multiple replies and provides a loaned container to access them.
- **dds::sub::LoanedSamples**< ReplyType > **take_replies** ()

  Takes all the replies.
- **dds::sub::LoanedSamples**< ReplyType > **take_replies** (const **rti::core::SampleIdentity** &related_request↩
_id)

Takes the replies for a specific requests.

- **dds::sub::LoanedSamples**< ReplyType > **read_replies** ()

  Reads all the replies.

- **dds::sub::LoanedSamples**< ReplyType > **read_replies** (const **rti::core::SampleIdentity** &related_request_id)

  Reads the replies for a specific request.

- bool **wait_for_replies** (const **dds::core::Duration** &max_wait)

  Waits for replies to any request.

- bool **wait_for_replies** (int min_count, const **dds::core::Duration** &max_wait)

  Waits for replies to any request.

- bool **wait_for_replies** (int min_count, const **dds::core::Duration** &max_wait, const **rti::core::SampleIdentity** &related_request_id)

  Waits for replies to a specific request.

- **dds::pub::DataWriter**< RequestType > **request_datawriter** () const

  *Retrieves the underlying **dds::pub::DataWriter** (p. 1138).*

- **dds::sub::DataReader**< ReplyType > **reply_datareader** () const

  *Retrieves the underlying **dds::sub::DataReader** (p. 976).*

Related Functions

(Note that these are not member functions.)

- template<typename RequestType , typename ReplyType >
  size_t **matched_replier_count** (**Requester**< RequestType, ReplyType > requester)

  *Obtains the number of Repliers that match with a **Requester** (p. 2187).*

8.313.1 Detailed Description

template<typename RequestType, typename ReplyType>
class rti::request::Requester< RequestType, ReplyType >

<<***reference-type***>> *(p. 351)* Allows sending requests and receiving replies

Note

A **Requester** (p. 2187) provides all the functions of a <<***reference-type***>> *(p. 351)* except **close()** (p. 1024) and retain().

A requester is an entity with two associated **topics** (p. 47): a request topic and a reply topic. It can send requests by publishing samples of the request topic and receive replies to those requests by subscribing to the reply topic.

Valid types for these topics are: those generated by rtiddsgen, the **DDS built-in types** (p. 50), and **dds::core::xtypes::DynamicData** (p. 1461).

A **Replier** (p. 2168) and a **Requester** (p. 2187) communicate when they use the same topics for requests and replies (see **RequesterParams::service_name** (p. 2200)) on the same domain ID.

A **Requester** (p. 2187) can send requests and receive one or multiple replies. It does that using the following operations:

- Sending requests (i.e. publishing request samples on the request topic)
- Waiting for replies to be received by the middleware (for any request or for a specific request)
- Getting those replies from the middleware. There are two ways to do this: take (the data samples are removed from the middleware), read (the data samples remain in the middleware and can be read or taken again).
- A convenience operation, receive (which is a combination of wait and take).

In all cases, the middleware guarantees that a requester only receives reply samples that are associated with those requests that it sends.

For multi-reply scenarios, in which a **Requester** (p. 2187) receives multiple replies from a **Replier** (p. 2168) for a given request, the **Requester** (p. 2187) can check if a reply is the last reply of a sequence of replies. To do so, see if the bit **rti::core::SampleFlag::intermediate_reply_sequence** (p. 2280) is set in **dds::sub::SampleInfo::flag** (p. 2293) after receiving each reply. This indicates it is NOT the last reply.

A requester has an associated **dds::domain::DomainParticipant** (p. 1326), which can be shared with other requesters or RTI Connext routines. All the other RTI Connext entities required for the request-reply interaction, including a **dds::pub::DataWriter** (p. 1138) for writing requests and a **dds::sub::DataReader** (p. 976) for reading replies, are automatically created when the requester is constructed.

Quality of Service for the underlying DataWriter and DataReader can be configured (see RequesterParams::qos_profile). By default, they are created with **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157). The exact default configuration is described here: **Configuring Request-Reply QoS profiles** (p. 348)

Template Parameters

RequestType	The data type for the request topic
ReplyType	The data type for the reply topic

See also

rti::request::Replier (p. 2168)

Request-Reply Examples (p. 344)

Basic Requester example (p. 346)

8.313.2 Constructor & Destructor Documentation

8.313.2.1 Requester() [1/2]

```
template<typename RequestType , typename ReplyType >
rti::request::Requester< RequestType, ReplyType >::Requester (
            const  RequesterParams & params )  [inline], [explicit]
```

Creates a **Requester** (p. 2187) with parameters.

Parameters

params	All the parameters that configure this requester. See **RequesterParams** (p. 2199) for the list of mandatory parameters.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

RequesterParams (p. 2199)

Requester Creation (p. 345)

8.313.2.2 Requester() [2/2]

```
template<typename RequestType , typename ReplyType >
rti::request::Requester< RequestType, ReplyType >::Requester (
              dds::domain::DomainParticipant participant,
             const std::string & service_name )  [inline]
```

Creates a **Requester** (p. 2187) with the minimum set of parameters.

Parameters

participant	The DomainParticipant this requester uses to join a DDS domain
service_name	The service name. See **RequesterParams::service_name** (p. 2200)

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.313.3 Member Function Documentation

8.313.3.1 wait_for_service()

```
template<typename RequestType , typename ReplyType >
bool  rti::request::Requester< RequestType, ReplyType >::wait_for_service (
             const  dds::core::Duration & max_wait =  dds::core::Duration::infinite() )  [inline]
```

Waits for one or more repliers to be discovered. This function takes advantage of enhanced discovery, making sure both the **Requester** (p. 2187) and the **Replier** (p. 2168) discover each other.

Parameters

max_wait	The maximum time to wait. Default: **dds::core::Duration::infinite()** (p. 1451) (unlimited period to wait).

Returns

True if a **Replier** (p. 2168) is discovered. False if a timeout occurs.

Exceptions

dds::core::AlreadyClosedError *(p. 806)*	if the **Requester** (p. 2187) is closed while waiting.

8.313.3.2 send_request() [1/2]

```
template<typename RequestType , typename ReplyType >
rti::core::SampleIdentity  rti::request::Requester< RequestType, ReplyType >::send_request (
            const RequestType & request )  [inline]
```

Sends a request.

Parameters

request	The request to be sent

Returns

The identity of the request, which can be used to correlate it with a future reply (see **rti::request::Requester↩ ::wait_for_replies(int, const dds::core::Duration&, const rti::core::SampleIdentity&)** (p. 2196))

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if a **Replier** (p. 2168) hasn't been discovered. Make sure to call **wait_for_service()** (p. 2190) before sending a request. (This exception is not thrown if this **Requester** (p. 2187) was created with **RequesterParams::require_matching_service_on_↩ send_request** (p. 2202) set to false.)
standard-dds-exception	One of the standard DDS exceptions (**Exceptions** (p. 427)).

8.313.3.3 send_request() [2/2]

```
template<typename RequestType , typename ReplyType >
void  rti::request::Requester< RequestType, ReplyType >::send_request (
            const RequestType & request,
             rti::pub::WriteParams & params )  [inline]
```

Sends a request with advanced parameters.

Parameters

request	The request to be sent
params	(in-out) The parameters to write the request (see **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179))

This function allows setting the request identity in `params.identity()`, among other advanced parameters. If the identity is not set, RTI Connext automatically assigns it; for the automatic assignment to occur, `params.replace_automatic_values()` needs to be set to true. After this function ends, `params.identity()` will contain the request identity. If no additional parameters need to be set, use the simpler **send_request(const RequestType&)** (p. 2191), which directly returns the automatically assigned identity.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if a **Replier** (p. 2168) hasn't been discovered. Make sure to call **wait_for_service()** (p. 2190) before sending a request. (This exception is not thrown if this **Requester** (p. 2187) was created with **RequesterParams::require_matching_service_on_send_request** (p. 2202) set to false.)
standard-dds-exception	One of the standard DDS exceptions (**Exceptions** (p. 427)).

8.313.3.4 receive_replies() [1/2]

```
template<typename RequestType , typename ReplyType >
dds::sub::LoanedSamples< ReplyType >  rti::request::Requester< RequestType, ReplyType >::receive_replies (
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for multiple replies and provides a loaned container to access them.

This operation is equivalent to using **rti::request::Requester::receive_replies** (p. 2192) with `min_count=1` and `max_count=dds::core::LENGTH_UNLIMITED` (p. 438)

MT Safety:

See **rti::request::Requester::wait_for_replies(int, const dds::core::Duration&)** (p. 2195)

See also

dds::sub::LoanedSamples (p. 1663)

rti::request::Requester::receive_replies (p. 2192)

8.313.3.5 receive_replies() [2/2]

```
template<typename RequestType , typename ReplyType >
dds::sub::LoanedSamples< ReplyType > rti::request::Requester< RequestType, ReplyType >::receive_replies (
            int min_count,
            const dds::core::Duration & max_wait ) [inline]
```

Waits for multiple replies and provides a loaned container to access them.

This operation is equivalent to using **rti::request::Requester::wait_for_replies(int, const dds::core::Duration&)** (p. 2195) and **rti::request::Requester::take_replies** (p. 2193).

MT Safety:

See **rti::request::Requester::wait_for_replies(int, const dds::core::Duration&)** (p. 2195)

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

dds::sub::LoanedSamples (p. 1663)

rti::request::Requester::wait_for_replies(int, const dds::core::Duration&) (p. 2195)

rti::request::Requester::take_replies (p. 2193)

8.313.3.6 take_replies() [1/2]

```
template<typename RequestType , typename ReplyType >
dds::sub::LoanedSamples< ReplyType > rti::request::Requester< RequestType, ReplyType >::take_replies ( ) [inline]
```

Takes all the replies.

Takes all the existing replies up to `max_count` and provides a loaned container to access them.

This operation does not make a copy of the data. It is similar to **dds::sub::DataReader::take** (p. 997).

The loan is returned with **dds::sub::LoanedSamples::return_loan** (p. 1667) or in the destructor

This operation may be used after a call to **rti::request::Requester::wait_for_replies(int, const dds::core::Duration&)** (p. 2195)

Returns

A container with up to max_count elements. May be empty if there were no replies to get.

Exceptions

One	of the **Standard Exceptions** (p. 428)

MT Safety:

SAFE

See also

dds::sub::LoanedSamples (p. 1663)

dds::sub::LoanedSamples::return_loan (p. 1667)

rti::request::Requester::wait_for_replies(int, const dds::core::Duration&) (p. 2195)

dds::sub::DataReader::take (p. 997) (for a more detailed description on how QoS and other **parameters** (p. 2063) affect the underlying DataReader)

8.313.3.7 take_replies() [2/2]

```
template<typename RequestType , typename ReplyType >
dds::sub::LoanedSamples< ReplyType >  rti::request::Requester< RequestType, ReplyType >::take_↩
replies (
            const  rti::core::SampleIdentity & related_request_id )  [inline]
```

Takes the replies for a specific requests.

This operation is operation is analogous to **take_replies()** (p. 2193) but it only returns those replies associated with a specific request.

Parameters

related_request↩ _id	Identifies a request previously sent by this **Requester** (p. 2187)'s **send_request()** (p. 2191).

8.313.3.8 read_replies() [1/2]

```
template<typename RequestType , typename ReplyType >
dds::sub::LoanedSamples< ReplyType >  rti::request::Requester< RequestType, ReplyType >::read_↩
replies ( )  [inline]
```

Reads all the replies.

This operation is equivalent to **rti::request::Requester::take_replies** (p. 2193) except the replies remain in the **Requester** (p. 2187) and can be read or taken again.

8.313.3.9 read_replies() [2/2]

```
template<typename RequestType , typename ReplyType >
dds::sub::LoanedSamples< ReplyType >  rti::request::Requester< RequestType, ReplyType >::read_↩
replies (
            const  rti::core::SampleIdentity & related_request_id )  [inline]
```

Reads the replies for a specific request.

This operation is equivalent to **take_replies(const rti::core::SampleIdentity&)** (p. 2194), except the replies remain in the **Requester** (p. 2187) and can be read or taken again.

8.313.3.10 wait_for_replies() [1/3]

```
template<typename RequestType , typename ReplyType >
bool  rti::request::Requester< RequestType, ReplyType >::wait_for_replies (
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for replies to any request.

This operation is equivalent to using **rti::request::Requester::wait_for_replies(int, const dds::core::Duration&)** (p. 2195) with `min_count=1`.

See also

rti::request::Requester::wait_for_replies(int, const dds::core::Duration&) (p. 2195)

8.313.3.11 wait_for_replies() [2/3]

```
template<typename RequestType , typename ReplyType >
bool  rti::request::Requester< RequestType, ReplyType >::wait_for_replies (
            int min_count,
            const  dds::core::Duration & max_wait )  [inline]
```

Waits for replies to any request.

This operation waits for min_count requests to be available for up to max_wait .

If this operation is called several times but the available replies are not taken (with **rti::request::Requester::take_↩ replies** (p. 2193)), this operation may return immediately and will not wait for new replies. New replies may replace existing ones if they are not taken, depending on the **dds::core::policy::History** (p. 1602) used to configure this **Requester** (p. 2187).

Parameters

min_count	Minimum number of replies that need to be available for this operation to unblock.
max_wait	Maximum waiting time after which this operation unblocks, regardless of how many replies are available.

Returns

true if at least minCount replies were available before maxWait elapsed, or false otherwise.

MT Safety:

Concurrent calls to this operation on the same object are not allowed. However, waiting for replies for specific requests in parallel is supported (see **rti::request::Requester::wait_for_replies(int, const dds::core::Duration&, const rti::core::SampleIdentity&)** (p. 2196)).

See also

rti::request::Requester::take_replies (p. 2193)

8.313.3.12 wait_for_replies() [3/3]

```
template<typename RequestType , typename ReplyType >
bool  rti::request::Requester< RequestType, ReplyType >::wait_for_replies (
            int min_count,
            const  dds::core::Duration & max_wait,
            const  rti::core::SampleIdentity & related_request_id )  [inline]
```

Waits for replies to a specific request.

This operation is analogous to **rti::request::Requester::wait_for_replies(int, const dds::core::Duration&)** (p. 2195) except this operation waits for replies for a specific request.

Parameters

min_count	Minimum number of replies for the related request that need to be available for this operation to unblock.
max_wait	Maximum wait time after which this operation unblocks, regardless of how many replies are available.
related_request_id	The identity of a request previously sent by this **Requester** (p. 2187)

Returns

true if at least min_count replies for the related request were available before max_wait elapsed, or false otherwise.

Exceptions

One	of the **Standard Exceptions** (p. 428)

MT Safety:

SAFE

See also

rti::request::Requester::wait_for_replies(int, const dds::core::Duration&) (p. 2195)

8.313.3.13 request_datawriter()

```
template<typename RequestType , typename ReplyType >
dds::pub::DataWriter< RequestType >  rti::request::Requester< RequestType, ReplyType >::request_datawriter ( ) const  [inline]
```

Retrieves the underlying **dds::pub::DataWriter** (p. 1138).

Accessing the request DataWriter may be useful for a number of advanced use cases, such as:

- Finding matching subscriptions (e.g., Repliers)
- Setting a DataWriter listener
- Getting DataWriter protocol or cache statuses

MT Safety:

SAFE

See also

dds::pub::DataWriter (p. 1138)
dds::pub::DataWriter (p. 1138)
dds::pub::matched_subscriptions() (p. 648)
dds::pub::matched_subscription_data() (p. 650)
dds::pub::DataWriter::set_listener (p. 1169)
dds::pub::DataWriter::datawriter_protocol_status() (p. 1176)

Referenced by **rti::request::Requester< RequestType, ReplyType >::matched_replier_count()**.

8.313.3.14 reply_datareader()

```
template<typename RequestType , typename ReplyType >
dds::sub::DataReader< ReplyType >  rti::request::Requester< RequestType, ReplyType >::reply_↩
datareader ( ) const  [inline]
```

Retrieves the underlying **dds::sub::DataReader** (p. 976).

Accessing the reply DataReader may be useful for a number of advanced use cases, such as:

- Finding matching publications (e.g., Repliers)
- Setting a DataReader listener
- Getting DataReader protocol or cache statuses

MT Safety:

SAFE

See also

dds::sub::DataReader (p. 976)
dds::sub::DataReader (p. 976)
dds::sub::matched_publications (p. 668)
dds::sub::DataReader::matched_publication_data (p. 1033)
dds::sub::DataReader::set_listener (p. 1006)
dds::sub::DataReader::datareader_protocol_status (p. 1013)

Referenced by **rti::request::Requester< RequestType, ReplyType >::matched_replier_count()**.

8.313.4 Friends And Related Function Documentation

8.313.4.1 matched_replier_count()

```
template<typename RequestType , typename ReplyType >
size_t matched_replier_count (
              Requester< RequestType, ReplyType > requester )  [related]
```

Obtains the number of Repliers that match with a **Requester** (p. 2187).

Note

This is a standalone function in the namespace rti::request

For a **Replier** (p. 2168) to be included in the total count, the **Replier** (p. 2168)'s DataReader or DataWriter role name (in **rti::core::policy::EntityName** (p. 1530)) must not have been modified.

References **rti::request::Requester< RequestType, ReplyType >::reply_datareader()**, and **rti::request::Requester< RequestType, ReplyType >::request_datawriter()**.

8.314 rti::request::RequesterParams Class Reference

Contains the parameters for creating a **rti::request::Requester** (p. 2187).

`#include <RequesterParams.hpp>`

Inherits rti::request::detail::EntityParamsWithSetters< ActualEntity >.

Public Member Functions

- **RequesterParams** (**dds::domain::DomainParticipant** participant)
  *Creates a **RequesterParams** (p. 2199) with the parameters a **Requester** (p. 2187) always needs.*
- **RequesterParams** & **service_name** (const std::string &name)
  *The service name that Repliers and **Requester** (p. 2187) use to match and communicate.*
- **RequesterParams** & **request_topic_name** (const std::string &name)
  Sets a specific request topic name.
- **RequesterParams** & **reply_topic_name** (const std::string &name)
  Sets a specific reply topic name.
- **RequesterParams** & **datawriter_qos** (const **dds::core::optional**< **dds::pub::qos::DataWriterQos** > &qos)
  Sets the quality of service of the request DataWriter.
- **RequesterParams** & **datareader_qos** (const **dds::core::optional**< **dds::sub::qos::DataReaderQos** > &qos)
  Sets the quality of service of the request DataReader.
- **RequesterParams** & **publisher** (**dds::pub::Publisher** publisher)
  Sets a specific Publisher.
- **RequesterParams** & **subscriber** (**dds::sub::Subscriber** subscriber)
  Sets a specific Subscriber.
- **RequesterParams** & **request_type** (const **dds::core::optional**< **dds::core::xtypes::DynamicType** > &type)
  The request type, when DynamicData is used.
- **RequesterParams** & **reply_type** (const **dds::core::optional**< **dds::core::xtypes::DynamicType** > &type)
  The reply type, when DynamicData is used.
- **RequesterParams** & **require_matching_service_on_send_request** (bool value)
  *Sets whether send_request will fail if there is no matching **Replier** (p. 2168).*
- bool **require_matching_service_on_send_request** () const
  Getter (see setter with the same name)

8.314.1 Detailed Description

Contains the parameters for creating a **rti::request::Requester** (p. 2187).

See also

Creating a Requester with optional parameters (p. 345)

8.314.2 Constructor & Destructor Documentation

8.314.2.1 RequesterParams()

```
rti::request::RequesterParams::RequesterParams (
              dds::domain::DomainParticipant participant )  [inline], [explicit]
```

Creates a **RequesterParams** (p. 2199) with the parameters a **Requester** (p. 2187) always needs.

In addition to the participant , a **Requester** (p. 2187) needs either:

- A service name (**RequesterParams::service_name** (p. 2200)),
- Or custom topic names (**RequesterParams::request_topic_name** (p. 2200) and **RequesterParams::reply_↩ topic_name** (p. 2201))

When **dds::core::xtypes::DynamicData** (p. 1461) is used, **RequesterParams::request_type** (p. 2202) and/or **RequesterParams::reply_type** (p. 2202) are required as well.

The rest of the parameters that can be set in a **RequesterParams** (p. 2199) object are optional.

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) this requester uses to join a domain.

See also

Creating a Requester with optional parameters (p. 345)

8.314.3 Member Function Documentation

8.314.3.1 service_name()

```
RequesterParams & rti::request::RequesterParams::service_name (
            const std::string & name )
```

The service name that Repliers and **Requester** (p. 2187) use to match and communicate.

A **Requester** (p. 2187) and a **Replier** (p. 2168) need to be configured with the same topic names in order to match.

The service name is used to generate a request topic and a reply topic that the **Requester** (p. 2187) and **Replier** (p. 2168) will use to communicate. For example, the service name "MyService" will be used to create topics named "MyServiceRequest" and "MyServiceReply".

In some cases, the name of these topics is known beforehand or needs to be customized for another reason. The service name can be overridden by setting specific request and reply topic names using **RequesterParams::request↩ _topic_name** (p. 2200) and **RequesterParams::reply_topic_name** (p. 2201).

8.314.3.2 request_topic_name()

```
RequesterParams & rti::request::RequesterParams::request_topic_name (
            const std::string & name )
```

Sets a specific request topic name.

The specified topic name will be used, instead of allowing a topic name to be generated based on the **RequesterParams::service_name** (p. 2200). If that topic already exists, it will be reused.

8.314.3.3 reply_topic_name()

```
RequesterParams & rti::request::RequesterParams::reply_topic_name (
            const std::string & name )
```

Sets a specific reply topic name.

The specified topic name will be used, instead of allowing a topic name to be generated based on the **RequesterParams::service_name** (p. 2200). If that topic already exists, it will be reused.

8.314.3.4 datawriter_qos()

```
RequesterParams & rti::request::RequesterParams::datawriter_qos (
            const  dds::core::optional< dds::pub::qos::DataWriterQos > & qos )
```

Sets the quality of service of the request DataWriter.

See also

dds::core::QosProvider (p. 2025) to access Qos profiles.

8.314.3.5 datareader_qos()

```
RequesterParams & rti::request::RequesterParams::datareader_qos (
            const  dds::core::optional< dds::sub::qos::DataReaderQos > & qos )
```

Sets the quality of service of the request DataReader.

See also

dds::core::QosProvider (p. 2025) to access Qos profiles.

8.314.3.6 publisher()

```
RequesterParams & rti::request::RequesterParams::publisher (
            dds::pub::Publisher publisher )
```

Sets a specific Publisher.

By default, a **Requester** (p. 2187) uses the DomainParticipant's implicit Publisher. Sometimes a different Publisher may be needed, for example, to use non-default PublisherQos.

8.314.3.7 subscriber()

```
RequesterParams & rti::request::RequesterParams::subscriber (
            dds::sub::Subscriber subscriber )
```

Sets a specific Subscriber.

By default, a **Requester** (p. 2187) uses the DomainParticipant's implicit Subscriber. Sometimes a different Subscriber may be needed, for example, to use non-default SubscriberQos.

8.314.3.8 request_type()

```
RequesterParams & rti::request::RequesterParams::request_type (
            const  dds::core::optional< dds::core::xtypes::DynamicType > & type )
```

The request type, when DynamicData is used.

This is required when the **Requester** (p. 2187) RequestType is **dds::core::xtypes::DynamicData** (p. 1461). Otherwise this is ignored.

8.314.3.9 reply_type()

```
RequesterParams & rti::request::RequesterParams::reply_type (
            const  dds::core::optional< dds::core::xtypes::DynamicType > & type )
```

The reply type, when DynamicData is used.

This is required when the **Requester** (p. 2187) ReplyType is **dds::core::xtypes::DynamicData** (p. 1461). Otherwise this is ignored.

8.314.3.10 require_matching_service_on_send_request() [1/2]

```
RequesterParams & rti::request::RequesterParams::require_matching_service_on_send_request (
            bool value )  [inline]
```

Sets whether send_request will fail if there is no matching **Replier** (p. 2168).

When set to `true`, **Requester::send_request()** (p. 2191) will check if there is at least one matching **Replier** (p. 2168) and throw an exception if there isn't one. This prevents `send_request` to silently drop a request. When set to `false`, `send_request` will not perform this precondition check.

Note that you can use `wait_for_service()` regardless of this setting.

[default] true

8.314.3.11 require_matching_service_on_send_request() [2/2]

```
bool rti::request::RequesterParams::require_matching_service_on_send_request ( ) const  [inline]
```

Getter (see setter with the same name)

8.315 dds::core::policy::ResourceLimits Class Reference

Controls the memory usage of **dds::pub::DataWriter** (p. 1138) or a **dds::sub::DataReader** (p. 976).

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **ResourceLimits** ()

  Creates the default policy.
- **ResourceLimits** (int32_t the_max_samples, int32_t the_max_instances, int32_t the_max_samples_per_↩ instance)

  *Creates a **ResourceLimits** (p. 2203) object with the specified max_samples, max_instances, and max_samples_per instance and default values for the rest of the parameters.*
- **ResourceLimits** & **max_samples** (int32_t the_max_samples)

  Sets the maximum number of data samples that a DataWriter or a DataReader can manage across all instances.
- int32_t **max_samples** () const

  Getter (see setter with the same name)
- **ResourceLimits** & **max_instances** (int32_t the_max_instances)

  Sets the maximum number of instances that a DataWriter or a DataReader can manage.
- int32_t **max_instances** () const

  Getter (see setter with the same name)
- **ResourceLimits** & **max_samples_per_instance** (int32_t the_max_samples_per_instance)

  Sets the maximum number of data samples per instance that a DataWriter or a DataReader can manage.
- int32_t **max_samples_per_instance** () const

  Getter (see setter with the same name)
- **dds::core::policy::ResourceLimits** & **initial_samples** (int32_t the_initial_samples)

  *<<**extension**>> (p. 116) Sets the number of samples that a DataReader or a DataWriter will preallocate.*
- int32_t **initial_samples** () const

  Getter (see setter with the same name)
- **dds::core::policy::ResourceLimits** & **initial_instances** (int32_t the_initial_instances)

  *<<**extension**>> (p. 116) Sets the number of instances that a DataReader or a DataWriter will preallocate.*
- int32_t **initial_instances** () const

  Getter (see setter with the same name)
- **dds::core::policy::ResourceLimits** & **instance_hash_buckets** (int32_t the_instance_hash_buckets)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Sets the number of hash buckets for looking up instances*
- int32_t **instance_hash_buckets** () const

  Getter (see setter with the same name)

8.315.1 Detailed Description

Controls the memory usage of **dds::pub::DataWriter** (p. 1138) or a **dds::sub::DataReader** (p. 976).

Entity:

dds::topic::Topic (p. 2474), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Status:

dds::core::status::StatusMask::sample_rejected() (p. 2382), **dds::core::status::SampleRejectedStatus** (p. 2315)

Properties:

RxO (p. 514) = NO
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

8.315.2 Usage

This policy controls the resources that RTI Connext can use to meet the requirements imposed by the application and other QoS settings.

For the reliability protocol (and **dds::core::policy::Durability** (p. 1433)), this QoS policy determines the actual maximum queue size when the **dds::core::policy::History** (p. 1602) is set to **dds::core::policy::HistoryKind_def::KEEP_ALL** (p. 1607).

In general, this QoS policy is used to limit the amount of system memory that RTI Connext can allocate. For embedded real-time systems and safety-critical systems, pre-determination of maximum memory usage is often required. In addition, dynamic memory allocation could introduce non-deterministic latencies in time-critical paths.

This QoS policy can be set such that an entity does not dynamically allocate any more memory after its initialization phase.

If **dds::pub::DataWriter** (p. 1138) objects are communicating samples faster than they are ultimately taken by the **dds::sub::DataReader** (p. 976) objects, the middleware will eventually hit against some of the QoS-imposed resource limits. Note that this may occur when just a single **dds::sub::DataReader** (p. 976) cannot keep up with its corresponding **dds::pub::DataWriter** (p. 1138). The behavior in this case depends on the setting for the **RELIABILITY** (p. 544). If reliability is **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157), then RTI Connext is allowed to drop samples. If the reliability is **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157), RTI Connext will block the **dds::pub::DataWriter** (p. 1138) or discard the sample at the **dds::sub::DataReader** (p. 976) in order not to lose existing samples.

The constant **dds::core::LENGTH_UNLIMITED** (p. 438) may be used to indicate the absence of a particular limit. For example setting **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207) to **dds::core::LENGTH_UNLIMITED** (p. 438) will cause RTI Connext not to enforce this particular limit.

If these resource limits are not set sufficiently, under certain circumstances the **dds::pub::DataWriter** (p. 1138) may block on a **write()** (p. 1179) call even though the **dds::core::policy::History** (p. 1602) is **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607). To guarantee the writer does not block for **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607), make sure the resource limits are set such that:

```
max_samples >= max_instances * max_samples_per_instance
```

See also

dds::core::policy::Reliability (p. 2149)

dds::core::policy::History (p. 1602)

8.315.3 Consistency

The setting of **dds::core::policy::ResourceLimits::max_samples** (p. 2205) must be consistent with **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207). For these two values to be consistent, it must be true that **dds::core::policy::ResourceLimits::max_samples** (p. 2205) >= **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207). As described above, this limit will not be enforced if **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207) is set to **dds::core::LENGTH_UNLIMITED** (p. 438).

The setting of **RESOURCE_LIMITS** (p. 546) `max_samples_per_instance` must be consistent with the **HISTORY** (p. 533) `depth`. For these two QoS to be consistent, it must be true that *depth* <= *max_samples_per_instance*.

See also

dds::core::policy::History (p. 1602)

8.315.4 Constructor & Destructor Documentation

8.315.4.1 ResourceLimits() [1/2]

```
dds::core::policy::ResourceLimits::ResourceLimits ( )  [inline]
```

Creates the default policy.

8.315.4.2 ResourceLimits() [2/2]

```
dds::core::policy::ResourceLimits::ResourceLimits (
            int32_t the_max_samples,
            int32_t the_max_instances,
            int32_t the_max_samples_per_instance )  [inline]
```

Creates a **ResourceLimits** (p. 2203) object with the specified max_samples, max_instances, and max_samples_per instance and default values for the rest of the parameters.

8.315.5 Member Function Documentation

8.315.5.1 max_samples() [1/2]

```
ResourceLimits & dds::core::policy::ResourceLimits::max_samples (
            int32_t the_max_samples )  [inline]
```

Sets the maximum number of data samples that a DataWriter or a DataReader can manage across all instances.

Specifies the maximum number of data samples a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) can manage across all the instances associated with it.

For unkeyed types, this value has to be equal to max_samples_per_instance if max_samples_per_instance is not equal to **dds::core::LENGTH_UNLIMITED** (p. 438).

When batching is enabled, the maximum number of data samples a **dds::pub::DataWriter** (p. 1138) can manage will also be limited by **rti::core::policy::DataWriterResourceLimits::max_batches** (p. 1240).

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 100 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_samples, >= max_samples_per_instance, >= **rti::core::policy::DataReaderResourceLimits::max_samples_per_remote_writer** (p. 1087) or >= **rti::core::policy::RtpsReliableWriterProtocol::heartbeats_per_max_samples** (p. 2240)

For **dds::pub::qos::DataWriterQos** (p. 1226) max_samples >= **rti::core::policy::RtpsReliableWriterProtocol::heartbeats_per_max_samples** (p. 2240) in **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217) if batching is disabled.

Categories

> **Immutable** (p. 243), **Durability** (p. 184), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Data Availability** (p. 167)

8.315.5.2 max_samples() [2/2]

```
int32_t dds::core::policy::ResourceLimits::max_samples ( ) const  [inline]
```

Getter (see setter with the same name)

8.315.5.3 max_instances() [1/2]

```
ResourceLimits & dds::core::policy::ResourceLimits::max_instances (
            int32_t the_max_instances )  [inline]
```

Sets the maximum number of instances that a DataWriter or a DataReader can manage.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), >= initial_instances

Categories

> **Immutable** (p. 243), **Durability** (p. 184), **Instance Lifecycle** (p. 193), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Data Availability** (p. 167)

8.315.5.4 max_instances() [2/2]

```
int32_t dds::core::policy::ResourceLimits::max_instances ( ) const  [inline]
```

Getter (see setter with the same name)

8.315.5.5 max_samples_per_instance() [1/2]

```
ResourceLimits & dds::core::policy::ResourceLimits::max_samples_per_instance (
            int32_t the_max_samples_per_instance )  [inline]
```

Sets the maximum number of data samples per instance that a DataWriter or a DataReader can manage.

While an unkeyed type is logically considered as a single instance, for unkeyed types this value has to be equal to max_samples or **dds::core::LENGTH_UNLIMITED** (p. 438).

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 100 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), <= max_samples or **dds::core::LENGTH_↩ UNLIMITED** (p. 438), >= **dds::core::policy::History::depth** (p. 1605)

Categories

Immutable (p. 243), **Durability** (p. 184), **Instance Lifecycle** (p. 193), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204), **Data Availability** (p. 167)

8.315.5.6 max_samples_per_instance() [2/2]

```
int32_t dds::core::policy::ResourceLimits::max_samples_per_instance ( ) const  [inline]
```

Getter (see setter with the same name)

8.315.5.7 initial_samples() [1/2]

```
dds::core::policy::ResourceLimits & initial_samples (
            int32_t the_initial_samples )
```

<<***extension***>> *(p. 116)* Sets the number of samples that a DataReader or a DataWriter will preallocate.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Specifies the initial number of data samples a **dds::pub::DataWriter** (p. 1138) (or **dds::sub::DataReader** (p. 976)) will manage across all the instances associated with it.

[default] 32

[range] [1,100 million], <= max_samples

Categories

Immutable (p. 243), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204)

8.315.5.8 initial_samples() [2/2]

```
int32_t initial_samples ( ) const
```

Getter (see setter with the same name)

8.315.5.9 initial_instances() [1/2]

```
dds::core::policy::ResourceLimits & initial_instances (
            int32_t the_initial_instances )
```

<<***extension***>> *(*p. *116)* Sets the number of instances that a DataReader or a DataWriter will preallocate.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

[default] 32

[range] [1,1 million], <= max_instances

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193), **Sample Memory Management** (p. 230), **Middleware Memory Management** (p. 204)

8.315.5.10 initial_instances() [2/2]

```
int32_t initial_instances ( ) const
```

Getter (see setter with the same name)

8.315.5.11 instance_hash_buckets() [1/2]

```
dds::core::policy::ResourceLimits & instance_hash_buckets (
            int32_t the_instance_hash_buckets )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Sets the number of hash buckets for looking up instances

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The instance hash table facilitates instance lookup. A higher number of buckets decreases instance lookup time but increases the memory usage.

[default] 1 **[range]** [1,1 million]

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193)

8.315.5.12 instance_hash_buckets() [2/2]

```
int32_t instance_hash_buckets ( ) const
```

Getter (see setter with the same name)

8.316 rti::core::Result< T > Class Template Reference

A result from an operation that doesn't throw exceptions containing a return code and (if successful) a value.

```
#include <Result.hpp>
```

Public Member Functions

- **Result** ()=default

  *Creates a **Result** (p. 2209) object with a return code of DDS_RETCODE_ERROR and no value.*
- **Result** (const T &value)

  *Creates a **Result** (p. 2209) object with a return code of DDS_RETCODE_OK and a value that is copied.*
- **Result** (T &&value)

  *Creates a **Result** (p. 2209) object with a return code of DDS_RETCODE_OK and a value that is moved.*
- **Result** (DDS_ReturnCode_t retcode)

  *Creates a **Result** (p. 2209) object with a return code and no value.*
- const T & **get** () const &noexcept

  Gets the value without checking if the operation was successful.
- T & **get** () &noexcept

  Gets the value without checking if the operation was successful.
- DDS_ReturnCode_t **get_return_code** () const noexcept

  Gets the return code.
- bool **is_ok** () const noexcept

  Returns true if the operation was successful.
- const T & **get_if_ok** () const &

  Gets the value if the operation was successful, otherwise throws an exception.
- T & **get_if_ok** () &

  Gets the value if the operation was successful, otherwise throws an exception.
- T && **get_if_ok** () &&

  Gets the value if the operation was successful, otherwise throws an exception.
- void **throw_if_error** () const

  Throws an exception if the operation was not successful.

8.316.1 Detailed Description

template<typename T>
class rti::core::Result< T >

A result from an operation that doesn't throw exceptions containing a return code and (if successful) a value.

For some critical operations in the API a exception-throwing and a noexcept version of the same function are provided. The noexcept versions return a Result<T> object to indicate if the operation was successful or not and, if it was, provide the return value of type T.

Note that **Result** (p. 2209)<void> is specialized to not provide a value, only a return code.

8.316.2 Constructor & Destructor Documentation

8.316.2.1 Result() [1/4]

```
template<typename T >
rti::core::Result< T >::Result ( )  [default]
```

Creates a **Result** (p. 2209) object with a return code of DDS_RETCODE_ERROR and no value.

8.316.2.2 Result() [2/4]

```
template<typename T >
rti::core::Result< T >::Result (
            const T & value )  [inline], [explicit]
```

Creates a **Result** (p. 2209) object with a return code of DDS_RETCODE_OK and a value that is copied.

8.316.2.3 Result() [3/4]

```
template<typename T >
rti::core::Result< T >::Result (
            T && value )  [inline], [explicit]
```

Creates a **Result** (p. 2209) object with a return code of DDS_RETCODE_OK and a value that is moved.

8.316.2.4 Result() [4/4]

```
template<typename T >
rti::core::Result< T >::Result (
            DDS_ReturnCode_t retcode )  [inline], [explicit]
```

Creates a **Result** (p. 2209) object with a return code and no value.

8.316.3 Member Function Documentation

8.316.3.1 get() [1/2]

```
template<typename T >
const T &  rti::core::Result< T >::get ( ) const &  [inline], [noexcept]
```

Gets the value without checking if the operation was successful.

If the operation was not successful, the behavior is undefined; use **is_ok()** (p. 2212) first or use **get_if_ok()** (p. 2212) instead.

8.316.3.2 get() [2/2]

```
template<typename T >
T &  rti::core::Result< T >::get ( ) &  [inline], [noexcept]
```

Gets the value without checking if the operation was successful.

If the operation was not successful, the behavior is undefined; use **is_ok()** (p. 2212) first or use **get_if_ok()** (p. 2212) instead.

8.316.3.3 get_return_code()

```
template<typename T >
DDS_ReturnCode_t  rti::core::Result< T >::get_return_code ( ) const  [inline], [noexcept]
```

Gets the return code.

8.316.3.4 is_ok()

```
template<typename T >
bool  rti::core::Result< T >::is_ok ( ) const  [inline], [noexcept]
```

Returns true if the operation was successful.

This is equivalent to **get_return_code()** (p. 2212) == DDS_RETCODE_OK

8.316.3.5 get_if_ok() [1/3]

```
template<typename T >
const T &  rti::core::Result< T >::get_if_ok ( ) const &  [inline]
```

Gets the value if the operation was successful, otherwise throws an exception.

References **rti::core::Result< T >::throw_if_error()**.

8.316.3.6 get_if_ok() [2/3]

```
template<typename T >
T &  rti::core::Result< T >::get_if_ok ( ) &  [inline]
```

Gets the value if the operation was successful, otherwise throws an exception.

References **rti::core::Result< T >::throw_if_error()**.

8.316.3.7 get_if_ok() [3/3]

```
template<typename T >
T &&  rti::core::Result< T >::get_if_ok ( ) &&  [inline]
```

Gets the value if the operation was successful, otherwise throws an exception.

References **rti::core::Result< T >::throw_if_error()**.

8.316.3.8 throw_if_error()

```
template<typename T >
void  rti::core::Result< T >::throw_if_error ( ) const  [inline]
```

Throws an exception if the operation was not successful.

If the operation was successful, this function does nothing.

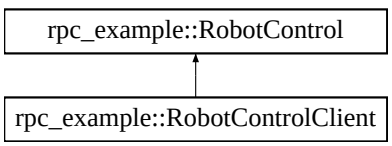
Referenced by **rti::core::Result< T >::get_if_ok()**.

8.317 rpc_example::RobotControl Class Reference

The synchronous interface generated from the **RobotControl** (p. 97) IDL service.

```
#include <ServiceEndpoint.hpp>
```

Inheritance diagram for rpc_example::RobotControl:



Public Member Functions

- virtual Coordinates **walk_to** (const Coordinates &destination, float speed)=0

  *An IDL interface method defined in **RobotControl** (p. 2213).*
- virtual float **get_speed** ()=0

  *An IDL interface method defined in **RobotControl** (p. 2213).*
- virtual ~**RobotControl** ()

  An empty virtual destructor.

8.317.1 Detailed Description

The synchronous interface generated from the **RobotControl** (p. 97) IDL service.

RobotControlClient (p. 2217) implements this interface to make synchronous remote function calls.

Server-side applications must implement this interface to handle remote calls. To execute an implementation of this interface, create a **RobotControlService** (p. 408).

8.317.2 Constructor & Destructor Documentation

8.317.2.1 ~RobotControl()

```
virtual rpc_example::RobotControl::~RobotControl ( )  [inline], [virtual]
```

An empty virtual destructor.

8.317.3 Member Function Documentation

8.317.3.1 walk_to()

```
virtual Coordinates rpc_example::RobotControl::walk_to (
            const Coordinates & destination,
            float speed )  [pure virtual]
```

An IDL interface method defined in **RobotControl** (p. 2213).

Parameters

destination	An input argument
speed	An input argument

Returns

A return value

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if a Service hasn't been discovered. Make sure to call **RobotControlClient::wait_for_service()** (p. 927) before sending a request.
dds::core::TimeoutError *(p. 2473)*	if a reply is not received after the timeout specified in the ClientParams.
dds::rpc::RemoteUnknownOperationError *(p. 2166)*	if the remote operation doesn't exist.
user-defined-exception	A user-defined exception (if any) when thrown on the server side.
standard-dds-exception	One of the standard DDS exceptions (**Exceptions** (p. 427)).

Note: the previous exceptions may be thrown on the client side.

8.317.3.2 get_speed()

```
virtual float rpc_example::RobotControl::get_speed ( )  [pure virtual]
```

An IDL interface method defined in **RobotControl** (p. 2213).

Returns

The return value of this method

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if a Service hasn't been discovered. Make sure to call **RobotControlClient::wait_for_service()** (p. 927) before sending a request.
dds::core::TimeoutError *(p. 2473)*	if a reply is not received after the timeout specified in the ClientParams.
dds::rpc::RemoteUnknownOperationError *(p. 2166)*	if the remote operation doesn't exist.
user-defined-exception	A user-defined exception (if any) when thrown on the server side.
standard-dds-exception	One of the standard DDS exceptions (**Exceptions** (p. 427)).

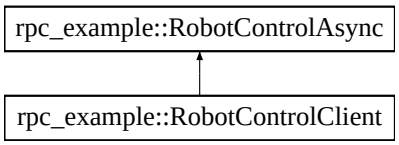
Note: the previous exceptions may be thrown on the client side.

8.318 rpc_example::RobotControlAsync Class Reference

The asynchronous interface derived from the **RobotControl** (p. 2213) service.

`#include <ServiceEndpoint.hpp>`

Inheritance diagram for rpc_example::RobotControlAsync:



Public Member Functions

- virtual std::future< Coordinates > **walk_to_async** (const Coordinates &destination, float speed)=0

  *An IDL interface method defined in **RobotControl** (p. 2213).*
- virtual std::future< float > **get_speed_async** ()=0

  *An IDL interface method defined in **RobotControl** (p. 2213).*
- virtual ~**RobotControlAsync** ()

  An empty virtual destructor.

8.318.1 Detailed Description

The asynchronous interface derived from the **RobotControl** (p. 2213) service.

RobotControlClient (p. 2217) implements this interface to make asynchronous remote function calls.

This interface is not implemented on the server side.

8.318.2 Constructor & Destructor Documentation

8.318.2.1 ~RobotControlAsync()

```
virtual rpc_example::RobotControlAsync::~RobotControlAsync ( )  [inline], [virtual]
```

An empty virtual destructor.

8.318.3 Member Function Documentation

8.318.3.1 walk_to_async()

```
virtual std::future< Coordinates > rpc_example::RobotControlAsync::walk_to_async (
            const Coordinates & destination,
            float speed )  [pure virtual]
```

An IDL interface method defined in **RobotControl** (p. 2213).

Parameters

destination	An input argument
speed	An input argument

Returns

The future return value

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	if a Service hasn't been discovered. Make sure to call wait_for_service() before sending a request.

8.318.3.2 get_speed_async()

```
virtual std::future< float > rpc_example::RobotControlAsync::get_speed_async ( )  [pure virtual]
```

An IDL interface method defined in **RobotControl** (p. 2213).

Exceptions

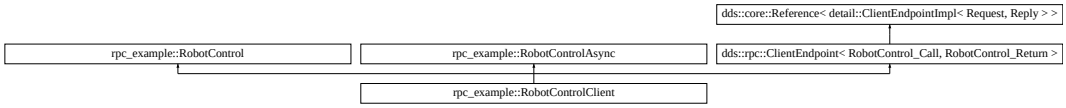
dds::core::PreconditionNotMetError *(p. 1941)*	if a Service hasn't been discovered. Make sure to call wait_for_service() before sending a request.

8.319 rpc_example::RobotControlClient Class Reference

<<***reference-type***>> *(p. 351)* Allows client applications to make remote function calls

```
#include <ClientEndpoint.hpp>
```

Inheritance diagram for rpc_example::RobotControlClient:



Additional Inherited Members

8.319.1 Detailed Description

<<***reference-type***>> *(*p. *351)* Allows client applications to make remote function calls

This class implements the **RobotControl** (p. 2213) and **RobotControlAsync** (p. 2215) interfaces to make remote function calls. It also inherits from RobotControlClientEndpoint, which manages the DDS entities required to make those function calls.

8.320 rti::config::compliance::RpcMask Class Reference

XTypes compliance mask.

`#include <Compliance.hpp>`

Inherits std::bitset< 32 >.

Public Types

- typedef std::bitset< 32 > **MaskType**
  A typedef of std::bitset<32> for convenience.

Public Member Functions

- **RpcMask** ()
  Default RPC compliance mask.
- **RpcMask** (uint64_t mask)
  *Construct a **RpcMask** (*p. *2218) from an integer.*
- **RpcMask** (const **MaskType** &mask)
  *Construct a **RpcMask** (*p. *2218) from a MaskType object.*

Static Public Member Functions

- static const **RpcMask use_standard_related_sample_pid** ()
  This bit configures a standard value for PID_RELATED_SAMPLE_IDENTITY.
- static const **RpcMask vendor** ()
  This mask configures compliance with the OMG DDS-RPC specification.
- static const **RpcMask default_mask** ()
  Default RPC compliance mask.

8.320.1 Detailed Description

XTypes compliance mask.

The **rti::config::compliance::XTypesMask** (p. 2666) configures different aspects of the Extended CDR encoding (XCDR and XCDR2).

Applications that don't use the exact same XTypes compliance mask may not be interoperable.

By default, data serialization is not fully compliant with Extended CDR encoding. While this doesn't affect correctness, it may prevent interoperability with other vendors.

You can change the default compliance mask for an application using any of the following mechanisms:

- Setting the environment variable **NDDS_XTYPES_COMPLIANCE_MASK**
- Calling the function **rti::config::compliance::set_xtypes_mask()** (p. 452)

The environment variable will be automatically loaded when a DomainParticipantFactory is created. If you need to load it before that, use the function **rti::config::compliance::load_compliance_masks()** (p. 453).

The format of the environment variable is an unsigned integer, which can be in HEX notation. For example: 0x00000001.

The default value is **rti::config::compliance::XTypesMask::default_mask()** (p. 2674).

To be fully compliant with the XTypes specification, set the mask to **rti::config::compliance::XTypesMask::vendor()** (p. 2674).

[default] rti::config::compliance::XTypesMask::default_mask() (p. 2674)

When changing the mask for an application is not desired, the property dds.xtypes.compliance_mask will change the mask at the DomainParticipant or DataWriter level. When set in the DomainParticipant, it configures the mask for all DataWriters in the participant. See `Extensible Types Compliance Mask` in the RTI Connext Core Libraries Extensible Types Guide.

8.320.2 Member Typedef Documentation

8.320.2.1 MaskType

`typedef std::bitset<32>` **`rti::config::compliance::RpcMask::MaskType`**

A typedef of std::bitset<32> for convenience.

8.320.3 Constructor & Destructor Documentation

8.320.3.1 RpcMask() [1/3]

```
rti::config::compliance::RpcMask::RpcMask ( )  [inline]
```

Default RPC compliance mask.

The mask does not sets any bits.

The HEX value for the mask is: 0x00000000.

8.320.3.2 RpcMask() [2/3]

```
rti::config::compliance::RpcMask::RpcMask (
            uint64_t mask )  [inline], [explicit]
```

Construct a **RpcMask** (p. 2218) from an integer.

Parameters

mask	Value whose bits are copied to the bitset positions

8.320.3.3 RpcMask() [3/3]

```
rti::config::compliance::RpcMask::RpcMask (
            const  MaskType & mask )  [inline]
```

Construct a **RpcMask** (p. 2218) from a MaskType object.

Parameters

mask	A std::bitset<32> to construct this **RpcMask** (p. 2218) from

8.320.4 Member Function Documentation

8.320.4.1 use_standard_related_sample_pid()

```
static const  RpcMask rti::config::compliance::RpcMask::use_standard_related_sample_pid ( )  [inline],
[static]
```

This bit configures a standard value for PID_RELATED_SAMPLE_IDENTITY.

Value: 0x00000001

Set this bit to be compliant with the OMG DDS-RPC specification. The PID_RELATED_SAMPLE_IDENTITY will be 0x0083 as is defined as is defined in the RPC over DDS OMG specification. See `https://www.omg.org/spec/DDS-RPC`

Using **rti::config::compliance::RpcMask::use_standard_related_sample_pid()** (p. 2220) will affect both RPC and Request-Reply communtications. The PID_RELATED_SAMPLE_IDENTITY is a parameter identifier (PID) used by the RTPS wire protocol and required for Request-Reply and RPC communications. By default, this PID takes a non-standard value that is compatible with previous versions of Connext.

This bit is unset by default. See **rti::config::compliance::RpcMask::default_mask()** (p. 2221)

8.320.4.2 vendor()

```
static const RpcMask rti::config::compliance::RpcMask::vendor ( ) [inline], [static]
```

This mask configures compliance with the OMG DDS-RPC specification.

The mask sets the following bits:

- **rti::config::compliance::RpcMask::use_standard_related_sample_pid()** (p. 2220)

The HEX value for the mask is: 0x00000001.

8.320.4.3 default_mask()

```
static const RpcMask rti::config::compliance::RpcMask::default_mask ( ) [inline], [static]
```

Default RPC compliance mask.

The mask does not sets any bits.

The HEX value for the mask is: 0x00000000.

8.321 rti::core::RtpsReliableReaderProtocol Class Reference

<<***extension***>> *(p. 354)* Configures aspects of the RTPS protocol related to a reliable DataReader

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **RtpsReliableReaderProtocol** ()

  Creates an instance with the default settings.
- **RtpsReliableReaderProtocol** & **min_heartbeat_response_delay** (const **dds::core::Duration** &the_min_↩ heartbeat_response_delay)

  <<***extension***>> *(*p. *116) The minimum delay to respond to a heartbeat.*
- **dds::core::Duration min_heartbeat_response_delay** () const

  Getter (see setter with the same name)
- **RtpsReliableReaderProtocol** & **max_heartbeat_response_delay** (const **dds::core::Duration** &the_max_↩ heartbeat_response_delay)

  <<***extension***>> *(*p. *116) The maximum delay to respond to a heartbeat.*
- **dds::core::Duration max_heartbeat_response_delay** () const

  Getter (see setter with the same name)
- **RtpsReliableReaderProtocol** & **heartbeat_suppression_duration** (const **dds::core::Duration** &the_↩ heartbeat_suppression_duration)

  <<***extension***>> *(*p. *116) The duration a reader ignores consecutively received heartbeats*
- **dds::core::Duration heartbeat_suppression_duration** () const

  Getter (see setter with the same name)
- **RtpsReliableReaderProtocol** & **nack_period** (const **dds::core::Duration** &the_nack_period)

  <<***extension***>> *(*p. *116) The period at which to send NACKs*
- **dds::core::Duration nack_period** () const

  Getter (see setter with the same name)
- **RtpsReliableReaderProtocol** & **receive_window_size** (int32_t the_receive_window_size)

  <<***extension***>> *(*p. *116) The number of received out-of-order samples a reader can keep at a time*
- int32_t **receive_window_size** () const

  Getter (see setter with the same name)
- **RtpsReliableReaderProtocol** & **round_trip_time** (const **dds::core::Duration** &the_round_trip_time)

  <<***extension***>> *(*p. *116) The duration from sending a NACK to receiving a repair of a sample*
- **dds::core::Duration round_trip_time** () const

  Getter (see setter with the same name)
- **RtpsReliableReaderProtocol** & **app_ack_period** (const **dds::core::Duration** &the_app_ack_period)

  <<***extension***>> *(*p. *116) The period at which application-level acknowledgment messages are sent*
- **dds::core::Duration app_ack_period** () const

  Getter (see setter with the same name)
- **RtpsReliableReaderProtocol** & **min_app_ack_response_keep_duration** (const **dds::core::Duration** &the↩ _app_ack_period)

  <<***extension***>> *(*p. *116) Minimum duration for which application-level acknowledgment response data is kept*
- **dds::core::Duration min_app_ack_response_keep_duration** () const

  Getter (see setter with the same name)
- **RtpsReliableReaderProtocol** & **samples_per_app_ack** (int32_t the_samples_per_app_ack)

  <<***extension***>> *(*p. *116) The minimum number of samples acknowledged by one application-level acknowledgment message*
- int32_t **samples_per_app_ack** () const

  Getter (see setter with the same name)

8.321.1 Detailed Description

<<***extension***>> *(p. 354)* Configures aspects of the RTPS protocol related to a reliable DataReader

It is used to config reliable reader according to RTPS protocol.

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

QoS:

rti::core::policy::DataReaderProtocol (p. 1059) **rti::core::policy::DiscoveryConfig** (p. 1271)

8.321.2 Constructor & Destructor Documentation

8.321.2.1 RtpsReliableReaderProtocol()

```
rti::core::RtpsReliableReaderProtocol::RtpsReliableReaderProtocol ( )  [inline]
```

Creates an instance with the default settings.

8.321.3 Member Function Documentation

8.321.3.1 min_heartbeat_response_delay() [1/2]

```
RtpsReliableReaderProtocol & rti::core::RtpsReliableReaderProtocol::min_heartbeat_response_delay
(
        const  dds::core::Duration & the_min_heartbeat_response_delay )
```

<<***extension***>> *(p. 116)* The minimum delay to respond to a heartbeat.

When a reliable reader receives a heartbeat from a remote writer and finds out that it needs to send back an ACK/NACK message, the reader can choose to delay a while. This sets the value of the minimum delay.

[default] 0 seconds

[range] [0, 1 year], <= max_heartbeat_response_delay

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.321.3.2 min_heartbeat_response_delay() [2/2]

```
dds::core::Duration rti::core::RtpsReliableReaderProtocol::min_heartbeat_response_delay ( ) const
```

Getter (see setter with the same name)

8.321.3.3 max_heartbeat_response_delay() [1/2]

```
RtpsReliableReaderProtocol & rti::core::RtpsReliableReaderProtocol::max_heartbeat_response_delay
(
            const  dds::core::Duration & the_max_heartbeat_response_delay )
```

<<***extension***>> *(*p. *116)* The maximum delay to respond to a heartbeat.

When a reliable reader receives a heartbeat from a remote writer and finds out that it needs to send back an ACK/NACK message, the reader can choose to delay a while. This sets the value of maximum delay.

[default] The default value depends on the container policy:

- For **rti::core::policy::DataReaderProtocol::rtps_reliable_reader** (p. 1064): 0.5 seconds
- For **rti::core::policy::DiscoveryConfig::publication_reader** (p. 1286): 0 seconds
- For **rti::core::policy::DiscoveryConfig::subscription_reader** (p. 1288): 0 seconds
- For **rti::core::policy::DiscoveryConfig::participant_message_reader** (p. 1295): 0 seconds

[range] [0, 1 year], >= min_heartbeat_response_delay

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.321.3.4 max_heartbeat_response_delay() [2/2]

```
dds::core::Duration rti::core::RtpsReliableReaderProtocol::max_heartbeat_response_delay ( ) const
```

Getter (see setter with the same name)

8.321.3.5 heartbeat_suppression_duration() [1/2]

```
RtpsReliableReaderProtocol & rti::core::RtpsReliableReaderProtocol::heartbeat_suppression_↩
duration (
            const  dds::core::Duration & the_heartbeat_suppression_duration )
```

<<***extension***>> *(*p. *116)* The duration a reader ignores consecutively received heartbeats

When a reliable reader receives consecutive heartbeats within a short duration that will trigger redundant NACKs, the reader may ignore the latter heartbeat(s). This sets the duration during which additionally received heartbeats are suppressed.

[default] 0.0625 seconds

[range] [0, 1 year],

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.321.3.6 heartbeat_suppression_duration() [2/2]

```
dds::core::Duration rti::core::RtpsReliableReaderProtocol::heartbeat_suppression_duration ( )
const
```

Getter (see setter with the same name)

8.321.3.7 nack_period() [1/2]

```
RtpsReliableReaderProtocol & rti::core::RtpsReliableReaderProtocol::nack_period (
            const  dds::core::Duration & the_nack_period )
```

<<***extension***>> *(*p. *116)* The period at which to send NACKs

A reliable reader will send periodic NACKs at this rate when it first matches with a reliable writer. The reader will stop sending NACKs when it has received all available historical data from the writer.

[default] 5 seconds

[range] [1 nanosec, 1 year]

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.321.3.8 nack_period() [2/2]

```
dds::core::Duration rti::core::RtpsReliableReaderProtocol::nack_period ( ) const
```

Getter (see setter with the same name)

8.321.3.9 receive_window_size() [1/2]

```
RtpsReliableReaderProtocol & rti::core::RtpsReliableReaderProtocol::receive_window_size (
            int32_t the_receive_window_size )
```

<<***extension***>> *(p. 116)* The number of received out-of-order samples a reader can keep at a time

A reliable reader stores the out-of-order samples it receives until it can present them to the application in-order. The receive window is the maximum number of out-of-order samples that a reliable reader keeps at a given time. When the receive window is full, subsequently received out-of-order samples are dropped.

[default] 256

[range] [>= 1]

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.321.3.10 receive_window_size() [2/2]

```
int32_t rti::core::RtpsReliableReaderProtocol::receive_window_size ( ) const
```

Getter (see setter with the same name)

8.321.3.11 round_trip_time() [1/2]

```
RtpsReliableReaderProtocol & rti::core::RtpsReliableReaderProtocol::round_trip_time (
            const  dds::core::Duration & the_round_trip_time )
```

<<***extension***>> *(p. 116)* The duration from sending a NACK to receiving a repair of a sample

This round-trip time is an estimate of the time starting from when the reader sends a NACK for a specific sample to when it receives that sample. For each sample, the reader will not send a subsequent NACK for it until the round-trip time has passed, thus preventing inefficient redundant requests.

[default] 0 seconds

[range] [0 nanosec, 1 year]

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.321.3.12 round_trip_time() [2/2]

```
dds::core::Duration rti::core::RtpsReliableReaderProtocol::round_trip_time ( ) const
```

Getter (see setter with the same name)

8.321.3.13 app_ack_period() [1/2]

```
RtpsReliableReaderProtocol & rti::core::RtpsReliableReaderProtocol::app_ack_period (
            const  dds::core::Duration & the_app_ack_period )
```

<<***extension***>> *(*p. *116)* The period at which application-level acknowledgment messages are sent

A **dds::sub::DataReader** (p. 976) sends application-level acknowledgment messages to a **dds::pub::DataWriter** (p. 1138) at this periodic rate, and will continue sending until it receives a message from the **dds::pub::DataWriter** (p. 1138) that it has received and processed the acknowledgment and an AppAckConfirmation has been received by the **dds::sub::DataReader** (p. 976). Note: application-level acknowledgment messages can also be sent non-periodically, as determined by **rti::core::RtpsReliableReaderProtocol::samples_per_app_ack** (p. 2228).

[default] 5 seconds

[range] [1 nanosec, 1 year]

Categories

Immutable (p. 243), **Reliability** (p. 222), **Application Acknowledgement** (p. 158)

8.321.3.14 app_ack_period() [2/2]

```
dds::core::Duration rti::core::RtpsReliableReaderProtocol::app_ack_period ( ) const
```

Getter (see setter with the same name)

8.321.3.15 min_app_ack_response_keep_duration() [1/2]

```
RtpsReliableReaderProtocol & rti::core::RtpsReliableReaderProtocol::min_app_ack_response_keep_↩
duration (
            const  dds::core::Duration & the_app_ack_period )
```

<<***extension***>> *(*p. *116)* Minimum duration for which application-level acknowledgment response data is kept

The user-specified response data of an explicit application-level acknowledgment (called by **dds::sub::DataReader↩ ::acknowledge_sample** (p. 1015) or **dds::sub::DataReader::acknowledge_all** (p. 1013)) is cached by the **dds::sub↩ ::DataReader** (p. 976) for the purpose of reliably resending the data with the acknowledgment message. After this duration has passed from the time of the first acknowledgment, the response data is dropped from the cache and will not be resent with future acknowledgments for the corresponding sample(s).

[default] 0 sec

[range] [0 sec, 1 year]

Categories

Immutable (p. 243), **Reliability** (p. 222), **Application Acknowledgement** (p. 158)

8.321.3.16 min_app_ack_response_keep_duration() [2/2]

```
dds::core::Duration rti::core::RtpsReliableReaderProtocol::min_app_ack_response_keep_duration ( )
const
```

Getter (see setter with the same name)

8.321.3.17 samples_per_app_ack() [1/2]

```
RtpsReliableReaderProtocol & rti::core::RtpsReliableReaderProtocol::samples_per_app_ack (
            int32_t the_samples_per_app_ack )
```

<<***extension***>> *(*p. *116)* The minimum number of samples acknowledged by one application-level acknowledgment message

This setting applies only when **dds::core::policy::Reliability::acknowledgment_kind** (p. 2154) = **rti::core::policy↩ ::AcknowledgmentKind_def::APPLICATION_EXPLICIT** (p. 798) or **rti::core::policy::AcknowledgmentKind_def↩ ::APPLICATION_AUTO** (p. 798)

A **dds::sub::DataReader** (p. 976) will immediately send an application-level acknowledgment message when it has at least this many samples that have been acknowledged. It will not send an acknowledgment message until it has at least this many samples pending acknowledgment.

For example, calling **dds::sub::DataReader::acknowledge_sample** (p. 1015) this many times consecutively will trigger the sending of an acknowledgment message. Calling **dds::sub::DataReader::acknowledge_all** (p. 1013) may trigger the sending of an acknowledgment message, if at least this many samples are being acknowledged at once.

This is independent of the **rti::core::RtpsReliableReaderProtocol::app_ack_period** (p. 2227), where a **dds::sub::↩ DataReader** (p. 976) will send acknowledgement messages at the periodic rate regardless.

When this is set to **dds::core::LENGTH_UNLIMITED** (p. 438), then acknowledgement messages are sent only periodically, at the rate set by **rti::core::RtpsReliableReaderProtocol::app_ack_period** (p. 2227).

[default] 1

[range] [1, 1000000], or **dds::core::LENGTH_UNLIMITED** (p. 438)

Categories

Immutable (p. 243), **Reliability** (p. 222), **Application Acknowledgement** (p. 158)

8.321.3.18 samples_per_app_ack() [2/2]

```
int32_t rti::core::RtpsReliableReaderProtocol::samples_per_app_ack ( ) const
```

Getter (see setter with the same name)

8.322 rti::core::policy::RtpsReliableWriterProtocol Class Reference

<<***extension***>> *(*p. *116)* Configures aspects of an RTPS reliable writer

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **RtpsReliableWriterProtocol** ()

  Creates an instance with the default settings.
- **RtpsReliableWriterProtocol** & **low_watermark** (int32_t the_low_watermark)

  <<***extension***>> *(*p. *116) When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the* ***dds::core::status::StatusMask::reliable_writer_cache_changed()*** *(*p. *2385) is considered to have changed.*
- int32_t **low_watermark** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **high_watermark** (int32_t the_high_watermark)

  <<***extension***>> *(*p. *116) When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, the* ***dds::core::status::StatusMask::reliable_writer_cache_changed()*** *(*p. *2385) is considered to have changed.*
- int32_t **high_watermark** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **heartbeat_period** (const **dds::core::Duration** &the_heartbeat_period)

  <<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116) The period at which to send heartbeats.*
- **dds::core::Duration heartbeat_period** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **fast_heartbeat_period** (const **dds::core::Duration** &the_fast_heartbeat_↩ period)

  <<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116) An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.*
- **dds::core::Duration fast_heartbeat_period** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **late_joiner_heartbeat_period** (const **dds::core::Duration** &the_late_joiner↩ _heartbeat_period)

*<<**basic**>> (p. 104) <<**extension**>> (p. 116) An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.*

- **dds::core::Duration late_joiner_heartbeat_period** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **virtual_heartbeat_period** (const **dds::core::Duration** &the_virtual_↩ heartbeat_period)

  *<<**extension**>> (p. 116) The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.*

- **RtpsReliableWriterProtocol** & **samples_per_virtual_heartbeat** (int32_t the_samples_per_virtual_heartbeat)

  *<<**extension**>> (p. 116) The number of samples that a reliable writer has to publish before sending a virtual heartbeat.*

- int32_t **samples_per_virtual_heartbeat** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **max_heartbeat_retries** (int32_t the_max_heartbeat_retries)

  *<<**extension**>> (p. 116) The maximum number of periodic heartbeat retries before marking a remote reader as inactive.*

- int32_t **max_heartbeat_retries** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **inactivate_nonprogressing_readers** (bool the_inactivate_nonprogressing_↩ readers)

  *<<**extension**>> (p. 116) Whether to treat remote readers as inactive when their NACKs do not progress*

- bool **inactivate_nonprogressing_readers** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **heartbeats_per_max_samples** (int32_t the_heartbeats_per_max_samples)

  *<<**extension**>> (p. 116) The number of piggyback heartbeats sent per max send window.*

- int32_t **heartbeats_per_max_samples** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **min_nack_response_delay** (const **dds::core::Duration** &the_min_nack_↩ response_delay)

  *<<**extension**>> (p. 116) The minimum delay to respond to a NACK or NACK_FRAG.*

- **dds::core::Duration min_nack_response_delay** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **max_nack_response_delay** (const **dds::core::Duration** &the_max_nack_↩ response_delay)

  *<<**extension**>> (p. 116) The maximum delay to respond to a NACK or NACK_FRAG.*

- **dds::core::Duration max_nack_response_delay** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **nack_suppression_duration** (const **dds::core::Duration** &the_nack_↩ suppression_duration)

  *<<**extension**>> (p. 116) The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.*

- **dds::core::Duration nack_suppression_duration** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **max_bytes_per_nack_response** (int32_t the_max_bytes_per_nack_↩ response)

  *<<**extension**>> (p. 116) The maximum total message size when resending rejected samples or data fragments.*

- int32_t **max_bytes_per_nack_response** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **disable_positive_acks_min_sample_keep_duration** (const **dds::core::↩ Duration** &duration)

  *<<**extension**>> (p. 116) The minimum duration a sample is queued for ACK-disabled readers.*

- **dds::core::Duration disable_positive_acks_min_sample_keep_duration** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **disable_positive_acks_max_sample_keep_duration** (const **dds::core::Duration** &duration)

  *<<**extension**>> (p. 116) The maximum duration a sample is queued for ACK-disabled readers.*
- **dds::core::Duration disable_positive_acks_max_sample_keep_duration** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **disable_positive_acks_enable_adaptive_sample_keep_duration** (bool disable)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Enables dynamic adjustment of sample keep duration in response to congestion.*
- bool **disable_positive_acks_enable_adaptive_sample_keep_duration** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **disable_positive_acks_decrease_sample_keep_duration_factor** (int32_t factor)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Controls rate of contraction of dynamic sample keep duration.*
- int32_t **disable_positive_acks_decrease_sample_keep_duration_factor** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **disable_positive_acks_increase_sample_keep_duration_factor** (int32_t factor)

  ***[DEPRECATED]** (p. **296**) <<**extension**>> (p. 116) Controls rate of growth of dynamic sample keep duration.*
- int32_t **disable_positive_acks_increase_sample_keep_duration_factor** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **min_send_window_size** (int32_t the_min_send_window_size)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) Minimum size of send window of unacknowledged samples*
- int32_t **min_send_window_size** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **max_send_window_size** (int32_t the_max_send_window_size)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) Maximum size of send window of unacknowledged samples*
- int32_t **max_send_window_size** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **send_window_update_period** (const **dds::core::Duration** &the_send_window_update_period)

  *<<**extension**>> (p. 116) Period in which send window may be dynamically changed*
- **dds::core::Duration send_window_update_period** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **send_window_increase_factor** (int32_t the_send_window_increase_factor)

  *<<**extension**>> (p. 116) Increases send window size by this percentage when reacting dynamically to network conditions*
- int32_t **send_window_increase_factor** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **send_window_decrease_factor** (int32_t the_send_window_decrease_factor)

  *<<**extension**>> (p. 116) Decreases send window size by this percentage when reacting dynamically to network conditions*
- int32_t **send_window_decrease_factor** () const

  Getter (see setter with the same name)
- **RtpsReliableWriterProtocol** & **enable_multicast_periodic_heartbeat** (bool the_enable_multicast_periodic_heartbeat)

<<***extension***>> *(p. 116) Whether periodic heartbeat messages are sent over multicast*

- bool **enable_multicast_periodic_heartbeat** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **multicast_resend_threshold** (int32_t the_multicast_resend_threshold)

  <<***extension***>> *(p. 116) The minimum number of requesting readers needed to trigger a multicast resend*

- int32_t **multicast_resend_threshold** () const

  Getter (see setter with the same name)

- **RtpsReliableWriterProtocol** & **disable_repair_piggyback_heartbeat** (bool the_disable_repair_piggyback_↩ heartbeat)

  <<***extension***>> *(p. 116) Prevents piggyback heartbeats from being sent with repair samples*

- bool **disable_repair_piggyback_heartbeat** () const

  Getter (see setter with the same name)

8.322.1 Detailed Description

<<***extension***>> *(p. 116)* Configures aspects of an RTPS reliable writer

It is used to configure a reliable writer according to RTPS protocol.

The reliability protocol settings are applied to batches instead of individual data samples when batching is enabled.

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

QoS:

rti::core::policy::DataWriterProtocol (p. 1210) **rti::core::policy::DiscoveryConfig** (p. 1271)

8.322.2 Constructor & Destructor Documentation

8.322.2.1 RtpsReliableWriterProtocol()

```
rti::core::policy::RtpsReliableWriterProtocol::RtpsReliableWriterProtocol ( )  [inline]
```

Creates an instance with the default settings.

8.322.3 Member Function Documentation

8.322.3.1 low_watermark() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::low_watermark (
            int32_t the_low_watermark )
```

<<***extension***>> *(*p. *116)* When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385) is considered to have changed.

This value is measured in units of samples, except with batching configurations where it is measured in units of batches.

The value must be greater than or equal to zero and strictly less than high_watermark.

The high and low watermarks are used for switching between the regular and fast heartbeat rates (**rti::core::policy::RtpsReliableWriterProtocol::heartbeat_period** (p. 2234) and **rti::core::policy::RtpsReliableWriterProtocol::fast_heartbeat_period** (p. 2235), respectively). When the number of unacknowledged samples in the queue of a reliable **dds::pub::DataWriter** (p. 1138) meets or exceeds high_watermark, the **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385) is changed, and the DataWriter will start heartbeating at **rti::core::policy::RtpsReliableWriterProtocol::fast_heartbeat_period** (p. 2235). When the number of samples meets or falls below low_watermark, **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385) is changed, and the heartbeat rate will return to the "normal" rate (**rti::core::policy::RtpsReliableWriterProtocol::heartbeat_period** (p. 2234)).

[default] 0

[range] [0, 100 million], < high_watermark

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.322.3.2 low_watermark() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::low_watermark ( ) const
```

Getter (see setter with the same name)

8.322.3.3 high_watermark() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::high_watermark (
            int32_t the_high_watermark )
```

<<***extension***>> *(*p. *116)* When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, the **dds::core::status::StatusMask::reliable_writer_cache_changed()** (p. 2385) is considered to have changed.

This value is measured in units of samples, except with batching configurations where it is measured in units of batches.

The value must be strictly greater than low_watermark and less than or equal to a maximum that depends on the container QoS policy:

In dds::domain::qos::DomainParticipantQos::discovery_config:

For **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289)
high_watermark<= **rti::core::AllocationSettings::max_count** (p. 804) in **rti::core::policy::DomainParticipant↩ ResourceLimits::local_writer_allocation** (p. 1398)

For **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291)
high_watermark<= **rti::core::AllocationSettings::max_count** (p. 804) in **rti::core::policy::DomainParticipant↩ ResourceLimits::local_reader_allocation** (p. 1398)

In **rti::core::policy::DataWriterProtocol** (p. 1210):

For **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217),

high_watermark<=**dds::core::policy::ResourceLimits::max_samples** (p. 2205) if batching is disabled. Otherwise, high_watermark<=**rti::core::policy::DataWriterResourceLimits::max_batches** (p. 1240) high_watermark<=**rti↩ ::core::policy::RtpsReliableWriterProtocol::max_send_window_size** (p. 2250)

[default] 1

[range] [1, 100 million] or **dds::core::LENGTH_UNLIMITED** (p. 438), > low_watermark <= *maximum* which depends on the container policy

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.322.3.4 high_watermark() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::high_watermark ( ) const
```

Getter (see setter with the same name)

8.322.3.5 heartbeat_period() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::heartbeat_period (
            const  dds::core::Duration & the_heartbeat_period )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* The period at which to send heartbeats.

A reliable writer will send periodic heartbeats at this rate.

[default] The default value depends on the container policy:

- For **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217): 200 ms
- For **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289): 3.0 seconds
- For **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291): 3.0 seconds
- For **rti::core::policy::DiscoveryConfig::participant_message_writer** (p. 1296): 1.0 seconds

[range] [1 nanosec,1 year], >= **rti::core::policy::RtpsReliableWriterProtocol::fast_heartbeat_period** (p. 2235), >= **rti::core::policy::RtpsReliableWriterProtocol::late_joiner_heartbeat_period** (p. 2236)

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.322.3.6 heartbeat_period() [2/2]

```
dds::core::Duration rti::core::policy::RtpsReliableWriterProtocol::heartbeat_period ( ) const
```

Getter (see setter with the same name)

8.322.3.7 fast_heartbeat_period() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::fast_heartbeat_period
(
            const  dds::core::Duration & the_fast_heartbeat_period )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.

This heartbeat period will be used when the number of unacknowledged samples in the cache of a reliable writer meets or exceeds the writer's high watermark and has not subsequently dropped to the low watermark. The normal period will be used at all other times.

This period must not be slower (i.e. must be of the same or shorter duration) than the normal heartbeat period.

[default] The default value depends on the container policy:

- For **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217): 200 ms seconds
- For **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289): 3.0 seconds
- For **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291): 3.0 seconds
- For **rti::core::policy::DiscoveryConfig::participant_message_writer** (p. 1296): 1.0 seconds

[range] [1 nanosec,1 year], <= **rti::core::policy::RtpsReliableWriterProtocol::heartbeat_period** (p. 2234)

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.322.3.8 fast_heartbeat_period() [2/2]

```
dds::core::Duration rti::core::policy::RtpsReliableWriterProtocol::fast_heartbeat_period ( )
const
```

Getter (see setter with the same name)

8.322.3.9 late_joiner_heartbeat_period() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::late_joiner_heartbeat↩
_period (
            const  dds::core::Duration & the_late_joiner_heartbeat_period )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.

This heartbeat period will be used when a reliable reader joins after a reliable writer with non-volatile durability has begun publishing samples. Once the reliable reader has received all cached samples, it will be serviced at the same rate as other reliable readers.

This period must not be slower (i.e., must be of the same or shorter duration) than the normal heartbeat period.

A reliable writer will use whichever heartbeat period is faster, the current heartbeat period being used for other reliable readers or the **rti::core::policy::RtpsReliableWriterProtocol::late_joiner_heartbeat_period** (p. 2236), to service the late joining reader. This means that if the **rti::core::policy::RtpsReliableWriterProtocol::fast_heartbeat_period** (p. 2235) is currently being used and is faster than the late_joiner_heartbeat_period, then the fast_heartbeat_period will continue to be used for the late joiner as well.

[default] The default value depends on the container policy:

- For **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217): 200 seconds
- For **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289): 3.0 seconds
- For **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291): 3.0 seconds
- For **rti::core::policy::DiscoveryConfig::participant_message_writer** (p. 1296): 1.0 seconds

[range] [1 nanosec,1 year], <= **rti::core::policy::RtpsReliableWriterProtocol::heartbeat_period** (p. 2234)

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.322.3.10 late_joiner_heartbeat_period() [2/2]

```
dds::core::Duration rti::core::policy::RtpsReliableWriterProtocol::late_joiner_heartbeat_period (
) const
```

Getter (see setter with the same name)

8.322.3.11 virtual_heartbeat_period()

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::virtual_heartbeat_↩
period (
            const  dds::core::Duration & the_virtual_heartbeat_period )
```

<<***extension***>> *(p. 116)* The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.

A reliable writer will send periodic virtual heartbeats at this rate.

[default] The default value depends on the container policy:

- For **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217): **dds::core::Duration::automatic()** (p. 1451). If **dds::core::policy::Presentation::access_scope** (p. 1947) is set to **dds::core::policy::↩ PresentationAccessScopeKind_def::GROUP** (p. 1951) on the DataWriter, this value is set to **rti::core↩ ::policy::RtpsReliableWriterProtocol::heartbeat_period** (p. 2234). Otherwise, the value is set to **dds::core↩ ::Duration::infinite()** (p. 1451).
- For **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289): **dds::core::Duration::infinite()** (p. 1451)
- For **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291): **dds::core::Duration::infinite()** (p. 1451)
- For **rti::core::policy::DiscoveryConfig::participant_message_writer** (p. 1296): **dds::core::Duration↩ ::infinite()** (p. 1451)

[range] $>$ 1 nanosec, **dds::core::Duration::infinite()** (p. 1451), or **dds::core::Duration::automatic()** (p. 1451)

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.322.3.12 samples_per_virtual_heartbeat() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::samples_per_virtual↩
_heartbeat (
            int32_t the_samples_per_virtual_heartbeat )
```

<<***extension***>> *(p. 116)* The number of samples that a reliable writer has to publish before sending a virtual heartbeat.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1,1000000], **dds::core::LENGTH_UNLIMITED** (p. 438)

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.322.3.13 samples_per_virtual_heartbeat() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::samples_per_virtual_heartbeat ( ) const
```

Getter (see setter with the same name)

8.322.3.14 max_heartbeat_retries() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::max_heartbeat_retries
(
            int32_t the_max_heartbeat_retries )
```

<<***extension***>> *(*p. *116)* The maximum number of *periodic* heartbeat retries before marking a remote reader as inactive.

When a remote reader has not acked all the samples the reliable writer has in its queue, and max_heartbeat_retries number of periodic heartbeats has been sent without receiving any ack/nack back, the remote reader will be marked as inactive (not alive) and be ignored until it resumes sending ack/nack.

Note that piggyback heartbeats do NOT count towards this value.

[default] 150

[range] [1, 1 million] or **dds::core::LENGTH_UNLIMITED** (p. 438)

Categories

Immutable (p. 243), **Reliability** (p. 222), **Liveliness** (p. 199)

8.322.3.15 max_heartbeat_retries() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::max_heartbeat_retries ( ) const
```

Getter (see setter with the same name)

8.322.3.16 inactivate_nonprogressing_readers() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::inactivate_nonprogressing↩
_readers (
            bool the_inactivate_nonprogressing_readers )
```

<<***extension***>> *(*p. *116)* Whether to treat remote readers as inactive when their NACKs do not progress

Nominally, a remote reader is marked inactive when a successive number of periodic heartbeats equal or greater than **rti::core::policy::RtpsReliableWriterProtocol::max_heartbeat_retries** (p. 2239) have been sent without receiving any ack/nacks back.

By setting this true, it changes the conditions of inactivating a remote reader: a reader will be considered inactive when it either does not send any ack/nacks or keeps sending non-progressing nacks for **rti::core::policy::RtpsReliable↩ WriterProtocol::max_heartbeat_retries** (p. 2239) number of heartbeat periods, where a non-progressing nack is one whose oldest sample requested has not advanced from the oldest sample requested of the previous nack.

[default] false

Categories

Immutable (p. 243), **Reliability** (p. 222), **Liveliness** (p. 199)

8.322.3.17 inactivate_nonprogressing_readers() [2/2]

```
bool rti::core::policy::RtpsReliableWriterProtocol::inactivate_nonprogressing_readers ( ) const
```

Getter (see setter with the same name)

8.322.3.18 heartbeats_per_max_samples() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::heartbeats_per_max_↩
samples (
            int32_t the_heartbeats_per_max_samples )
```

<<***extension***>> *(*p. *116)* The number of piggyback heartbeats sent per max send window.

When a DataWriter is configured with a fixed send window size (**rti::core::policy::RtpsReliableWriterProtocol::min↩ _send_window_size** (p. 2249) is equal to effective max_send_window_size), a piggyback heartbeat is sent every [(effective max send window size/heartbeats_per_max_samples)] number of samples written.

Otherwise, the number of piggyback heartbeats sent is scaled according to the current size of the send window. For example, consider a **rti::core::policy::RtpsReliableWriterProtocol::heartbeats_per_max_samples** (p. 2240) of 50. If the current send window size is 100, a piggyback heartbeat will be sent every 2 samples. If the send window size grows to 150, a piggyback heartbeat will be sent every 3 samples, and so on. Additionally, when the send window size

grows, a piggyback heartbeat is sent with the next sample. (If it weren't, the sending of that heartbeat could be delayed, since the heartbeat rate scales with the increasing window size.)

The effective max send window is calculated as follows:

Without batching:

min (**dds::core::policy::ResourceLimits::max_samples** (p. 2205), **rti::core::policy::RtpsReliableWriterProtocol↩ ::max_send_window_size** (p. 2250))

With batching:

min (**rti::core::policy::DataWriterResourceLimits::max_batches** (p. 1240), **rti::core::policy::RtpsReliableWriter↩ Protocol::max_send_window_size** (p. 2250))

If heartbeats_per_max_samples is set to zero, no piggyback heartbeats will be sent.

If current send window size is **dds::core::LENGTH_UNLIMITED** (p. 438), 100 million is assumed as the effective max send window.

[default] The default value depends on the container policy:

- For **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217): 8
- For **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289): 8
- For **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291): 8
- For **rti::core::policy::DiscoveryConfig::participant_message_writer** (p. 1296): 1

[range] [0, 100 million]

- For **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289): heartbeats_per_max_samples<= **rti::core::AllocationSettings::max_count** (p. 804) in **rti::core::policy::↩ DomainParticipantResourceLimits::local_writer_allocation** (p. 1398)
- For **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291): heartbeats_per_max_samples<= **rti::core::AllocationSettings::max_count** (p. 804) in **rti::core::policy::↩ DomainParticipantResourceLimits::local_reader_allocation** (p. 1398)
- For **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217):

heartbeats_per_max_samples<= **dds::core::policy::ResourceLimits::max_samples** (p. 2205) if batching is disabled. Otherwise:
heartbeats_per_max_samples<= **rti::core::policy::DataWriterResourceLimits::max_batches** (p. 1240)

heartbeats_per_max_samples<= **rti::core::policy::RtpsReliableWriterProtocol::max_send_window_size** (p. 2250)

Categories

Immutable (p. 243), **Reliability** (p. 222)

8.322.3.19 heartbeats_per_max_samples() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::heartbeats_per_max_samples ( ) const
```

Getter (see setter with the same name)

8.322.3.20 min_nack_response_delay() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::min_nack_response_↩
delay (
            const  dds::core::Duration & the_min_nack_response_delay )
```

<<***extension***>> *(p. 116)* The minimum delay to respond to a NACK or NACK_FRAG.

When a reliable writer receives a NACK or a NACK_FRAG from a remote reader, the writer can choose to delay a while before it sends repair samples/fragments or a heartbeat. This sets the value of the minimum delay.

Delaying repairs can lower the amount of duplicate repair data sent to the remote reader. If the writer receives the same NACKs or NACK_FRAGs back-to-back within this wait time, it will combine them into one repair action.

[default] 0 seconds

[range] [0, 1 day], <= max_nack_response_delay

Categories

Immutable (p. 243), **Reliability** (p. 222), **Fragmentation** (p. 191)

8.322.3.21 min_nack_response_delay() [2/2]

```
dds::core::Duration rti::core::policy::RtpsReliableWriterProtocol::min_nack_response_delay ( )
const
```

Getter (see setter with the same name)

8.322.3.22 max_nack_response_delay() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::max_nack_response_↩
delay (
            const  dds::core::Duration & the_max_nack_response_delay )
```

<<***extension***>> *(p. 116)* The maximum delay to respond to a NACK or NACK_FRAG.

This sets the value of maximum delay between receiving a NACK or a NACK_FRAG and sending repair samples/fragments or a heartbeat.

[default] The default value depends on the container policy:

- For **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217): 0.2 seconds
- For **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289): 0 seconds
- For **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291): 0 seconds
- For **rti::core::policy::DiscoveryConfig::participant_message_writer** (p. 1296): 0 seconds

[range] [0,1 day], >= min_nack_response_delay

Categories

Immutable (p. 243), **Reliability** (p. 222), **Fragmentation** (p. 191)

8.322.3.23 max_nack_response_delay() [2/2]

```
dds::core::Duration rti::core::policy::RtpsReliableWriterProtocol::max_nack_response_delay ( )
const
```

Getter (see setter with the same name)

8.322.3.24 nack_suppression_duration() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::nack_suppression_↩
duration (
            const  dds::core::Duration & the_nack_suppression_duration )
```

<<***extension***>> *(p. 116)* The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.

A reliable writer may receive consecutive NACKs/NACK_FRAGs within a short duration from a remote reader that will trigger the sending of redundant repair messages.

This specifies the duration during which consecutive NACKs/NACK_FRAGs are ignored to prevent redundant repairs from being sent.

A redundant NACK is a NACK that requests the same or lower first sequence number (SN) than the last NACK from the same reader.

A redundant NACK_FRAG is a NACK_FRAG that requests the same first fragment number for a sample as the last NACK_FRAG from the same reader.

The suppression duration applies separately to NACKs and NACK_FRAGS. For example, replying to a NACK_FRAG message in between receiving two NACK messages will not cause the second NACK message to be suppressed if the two NACK messages are received more than the suppression duration apart.

[default] 0 seconds

[range] [0, 1 day]

Categories

Immutable (p. 243), **Reliability** (p. 222), **Fragmentation** (p. 191)

8.322.3.25 nack_suppression_duration() [2/2]

```
dds::core::Duration rti::core::policy::RtpsReliableWriterProtocol::nack_suppression_duration ( )
const
```

Getter (see setter with the same name)

8.322.3.26 max_bytes_per_nack_response() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::max_bytes_per_nack_↩
response (
            int32_t the_max_bytes_per_nack_response )
```

<<***extension***>> *(*p. *116)* The maximum total message size when resending rejected samples or data fragments.

As part of the reliable communication protocol, DataWriters send heartbeat (HB) messages to their DataReaders. Each HB message contains the sequence number of the most recent sample sent by the DataWriter.

In response, a DataReader may send a negative acknowledgement (NACK) message, indicating what sequence numbers it did not receive, if any. If the samples were fragmented, it can also send a NACK_FRAG requesting the fragments of a sample that did not arrive. If the DataReader is missing some samples or fragments, the DataWriter will send them again.

max_bytes_per_nack_response determines the maximum size of the message sent by the DataWriter in response to a NACK or NACK_FRAG. This message may contain multiple samples or fragments. The DataWriter will always send at least one message, even if the size of that message exceeds the max_bytes_per_nack_response value.

If the total size of all samples or fragments that needs to be resent is larger than max_bytes_per_nack_response, the remaining samples or fragments will be resent the next time a NACK or NACK_FRAG arrives.

Note: If **rti::core::policy::RtpsReliableWriterProtocol::min_nack_response_delay** (p. 2242) and **rti::core::policy↩ ::RtpsReliableWriterProtocol::max_nack_response_delay** (p. 2242) are both **dds::core::Duration::zero()** (p. 1450) (which means repairs are processed immediately), this limit applies separately to NACKs and NACK_FRAGs: repairing a NACK in this case does not affect the maximum size of a NACK_FRAG response and vice versa. Otherwise, the limit applies to the total size of the response, including received NACKs and NACK_FRAGs.

[default] The default value depends on the container policy:

- For **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217): 131072 bytes
- For **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289): 131072 bytes
- For **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291): 131072 bytes
- For **rti::core::policy::DiscoveryConfig::participant_message_writer** (p. 1296): 9216 bytes

[range] [0, 1 GB]

Categories

Immutable (p. 243), **Reliability** (p. 222), **Flow Control** (p. 188), **Fragmentation** (p. 191)

8.322.3.27 max_bytes_per_nack_response() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::max_bytes_per_nack_response ( ) const
```

Getter (see setter with the same name)

8.322.3.28 disable_positive_acks_min_sample_keep_duration() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::disable_positive_↩
acks_min_sample_keep_duration (
            const  dds::core::Duration & duration )
```

<<***extension***>> *(*p. *116)* The minimum duration a sample is queued for ACK-disabled readers.

When positive ACKs are disabled for a data writer (**rti::core::policy::DataWriterProtocol::disable_positive_acks** (p. 1214) = true) or a data reader (**rti::core::policy::DataReaderProtocol::disable_positive_acks** (p. 1062) = true), a sample is available from the data writer's queue for at least this duration, after which the sample may be considered to be acknowledged.

[default] 1 millisecond

[range] [0,1 year], <= **rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_max_sample_keep↩_duration** (p. 2246)

Categories

Immutable (p. 243), **Reliability** (p. 222), **Durability** (p. 184)

8.322.3.29 disable_positive_acks_min_sample_keep_duration() [2/2]

```
dds::core::Duration rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_min_↩
sample_keep_duration ( ) const
```

Getter (see setter with the same name)

8.322.3.30 disable_positive_acks_max_sample_keep_duration() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::disable_positive_↩
acks_max_sample_keep_duration (
            const  dds::core::Duration & duration )
```

<<***extension***>> *(*p. *116)* The maximum duration a sample is queued for ACK-disabled readers.

When positive ACKs are disabled for a data writer (**rti::core::policy::DataWriterProtocol::disable_positive_acks** (p. 1214) = true) or a data reader (**rti::core::policy::DataReaderProtocol::disable_positive_acks** (p. 1062) = true), a sample is available from the data writer's queue for at most this duration, after which the sample is considered to be acknowledged.

[default] 1 second

[range] [0,1 year], >= **rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_min_sample_keep↩_duration** (p. 2245)

Categories

Immutable (p. 243), **Reliability** (p. 222), **Durability** (p. 184)

8.322.3.31 disable_positive_acks_max_sample_keep_duration() [2/2]

```
dds::core::Duration rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_max_↩
sample_keep_duration ( ) const
```

Getter (see setter with the same name)

8.322.3.32 disable_positive_acks_enable_adaptive_sample_keep_duration() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::disable_positive_↩
acks_enable_adaptive_sample_keep_duration (
            bool disable )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Enables dynamic adjustment of sample keep duration in response to congestion.

For dynamic networks where a static minimum sample keep duration may not provide sufficient performance or reliability, setting **rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_enable_adaptive_sample_↩ keep_duration** (p. 2247) = true, enables the sample keep duration to be dynamically adjusted to adapt to network conditions. The keep duration changes according to the detected level of congestion, which is determined to be proportional to the rate of NACKs received. An adaptive algorithm automatically controls the keep duration to optimize throughput and reliability.

To relieve high congestion, the keep duration is increased to effectively decrease the send rate; this lengthening of the keep duration is controlled by **rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_increase↩ _sample_keep_duration_factor** (p. 2248). Alternatively, when congestion is low, the keep duration is decreased to effectively increase send rate; this shortening of the keep duration is controlled by **rti::core::policy::RtpsReliable↩ WriterProtocol::disable_positive_acks_decrease_sample_keep_duration_factor** (p. 2248).

The lower and upper bounds of the dynamic sample keep duration are set by **rti::core::policy::RtpsReliableWriter↩ Protocol::disable_positive_acks_min_sample_keep_duration** (p. 2245) and **rti::core::policy::RtpsReliable↩ WriterProtocol::disable_positive_acks_max_sample_keep_duration** (p. 2246), respectively.

When **rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_enable_adaptive_sample_keep↩ _duration** (p. 2247) = false, the sample keep duration is set to **rti::core::policy::RtpsReliableWriterProtocol↩ ::disable_positive_acks_min_sample_keep_duration** (p. 2245) .

[default] true

Categories

Immutable (p. 243), **Reliability** (p. 222), **Durability** (p. 184)

8.322.3.33 disable_positive_acks_enable_adaptive_sample_keep_duration() [2/2]

```
bool rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_enable_adaptive_sample↩
_keep_duration ( ) const
```

Getter (see setter with the same name)

8.322.3.34 disable_positive_acks_decrease_sample_keep_duration_factor() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::disable_positive_↩
acks_decrease_sample_keep_duration_factor (
            int32_t factor )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Controls rate of contraction of dynamic sample keep duration.

Used when **rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_enable_adaptive_sample_↩ keep_duration** (p. 2247) = true.

When the adaptive algorithm determines that the keep duration should be decreased, this factor (a percentage) is multiplied with the current keep duration to get the new shorter keep duration. For example, if the current keep duration is 20 milliseconds, using the default factor of 95% would result in a new keep duration of 19 milliseconds.

[default] 95

[range] <= 100

Categories

Immutable (p. 243), **Reliability** (p. 222), **Durability** (p. 184)

8.322.3.35 disable_positive_acks_decrease_sample_keep_duration_factor() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_decrease_sample_↩
keep_duration_factor ( ) const
```

Getter (see setter with the same name)

8.322.3.36 disable_positive_acks_increase_sample_keep_duration_factor() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::disable_positive_↩
acks_increase_sample_keep_duration_factor (
            int32_t factor )
```

[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Controls rate of growth of dynamic sample keep duration.

Used when **rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_enable_adaptive_sample_↩ keep_duration** (p. 2247) = true.

When the adaptive algorithm determines that the keep duration should be increased, this factor (a percentage) is multiplied with the current keep duration to get the new longer keep duration. For example, if the current keep duration is 20 milliseconds, using the default factor of 150% would result in a new keep duration of 30 milliseconds.

[default] 150

[range] >= 100

Categories

Immutable (p. 243), **Reliability** (p. 222), **Durability** (p. 184)

8.322.3.37 disable_positive_acks_increase_sample_keep_duration_factor() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::disable_positive_acks_increase_sample_↩
keep_duration_factor ( ) const
```

Getter (see setter with the same name)

8.322.3.38 min_send_window_size() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::min_send_window_size
(
            int32_t the_min_send_window_size )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Minimum size of send window of unacknowledged samples

A **dds::pub::DataWriter** (p. 1138) has a limit on the number of unacknowledged samples in-flight at a time. This send window can be configured to have a minimum size (this field) and a maximum size (max_send_window_size). The send window can dynamically change, between the min and max sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

When a variable sized send window is used (i.e., when min_send_window_size and max_send_window_size are not set to the same value) the send window is initialized to min_send_window_size.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] > 0, <= max_send_window_size, or **dds::core::LENGTH_UNLIMITED** (p. 438)

See also

rti::core::policy::RtpsReliableWriterProtocol::max_send_window_size (p. 2250)

rti::core::policy::RtpsReliableWriterProtocol::low_watermark (p. 2232)

rti::core::policy::RtpsReliableWriterProtocol::high_watermark (p. 2233)

rti::core::status::ReliableWriterCacheChangedStatus::full_reliable_writer_cache (p. 2160)

Categories

Immutable (p. 243), **Reliability** (p. 222), **Flow Control** (p. 188)

8.322.3.39 min_send_window_size() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::min_send_window_size ( ) const
```

Getter (see setter with the same name)

8.322.3.40 max_send_window_size() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::max_send_window_size
(
            int32_t the_max_send_window_size )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Maximum size of send window of unacknowledged samples

A **dds::pub::DataWriter** (p. 1138) has a limit on the number of unacknowledged samples in-flight at a time. This send window can be configured to have a minimum size (min_send_window_size) and a maximum size (this field). The send window can dynamically change, between the min and max sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

When a variable sized send window is used (i.e., when min_send_window_size and max_send_window_size are not set to the same value) the send window is initialized to min_send_window_size.

When both min_send_window_size and max_send_window_size are **dds::core::LENGTH_UNLIMITED** (p. 438), then either **dds::core::policy::ResourceLimits::max_samples** (p. 2205) (for non-batching) or **rti::core::policy::Data↩ WriterResourceLimits::max_batches** (p. 1240) (for batching) serves as the effective max_send_window_size. When **dds::core::policy::ResourceLimits::max_samples** (p. 2205) (for non-batching) or **rti::core::policy::DataWriter↩ ResourceLimits::max_batches** (p. 1240) (for batching) is less than max_send_window_size, then it serves as the effective max_send_window_size. If it is also less than min_send_window_size, then effectively both min and max send window sizes are equal to max_samples or max_batches.

In addition, the low and high watermarks are scaled down linearly to stay within the current send window size, and the full reliable queue status is set when the send window is full.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] $>$ 0, $>=$ min_send_window_size, or **dds::core::LENGTH_UNLIMITED** (p. 438)

See also

rti::core::policy::RtpsReliableWriterProtocol::min_send_window_size (p. 2249)

rti::core::policy::RtpsReliableWriterProtocol::low_watermark (p. 2232)

rti::core::policy::RtpsReliableWriterProtocol::high_watermark (p. 2233)

rti::core::status::ReliableWriterCacheChangedStatus::full_reliable_writer_cache (p. 2160)

Categories

Immutable (p. 243), **Reliability** (p. 222), **Flow Control** (p. 188)

8.322.3.41 max_send_window_size() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::max_send_window_size ( ) const
```

Getter (see setter with the same name)

8.322.3.42 send_window_update_period() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::send_window_update_↩
period (
            const  dds::core::Duration & the_send_window_update_period )
```

<<***extension***>> *(*p. *116)* Period in which send window may be dynamically changed

The **dds::pub::DataWriter** (p. 1138)'s send window will dynamically change, between the min and max send window sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

The change in send window size happens at this update period, whereupon the send window is either increased or decreased in size according to the increase or decrease factors, respectively.

[default] The default value depends on the container policy:

- For **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217): 3 seconds
- For **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289): 3 seconds
- For **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291): 3 seconds
- For **rti::core::policy::DiscoveryConfig::participant_message_writer** (p. 1296): 1 second

[range] > [0,1 year]

See also

rti::core::policy::RtpsReliableWriterProtocol::send_window_increase_factor (p. 2252), **rti::core::policy::↩ RtpsReliableWriterProtocol::send_window_decrease_factor** (p. 2253)

Categories

Immutable (p. 243), **Reliability** (p. 222), **Flow Control** (p. 188)

8.322.3.43 send_window_update_period() [2/2]

```
dds::core::Duration rti::core::policy::RtpsReliableWriterProtocol::send_window_update_period ( )
const
```

Getter (see setter with the same name)

8.322.3.44 send_window_increase_factor() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::send_window_increase↩
_factor (
            int32_t the_send_window_increase_factor )
```

<<***extension***>> *(p. 116)* Increases send window size by this percentage when reacting dynamically to network conditions

The **dds::pub::DataWriter** (p. 1138)'s send window will dynamically change, between the min and max send window sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

After an update period during which no negative acknowledgements were received, the send window will be increased by this factor. The factor is treated as a percentage, where a factor of 150 would increase the send window by 150%. The increased send window size will not exceed the max_send_window_size.

[default] 105

[range] $>$ 100

See also

rti::core::policy::RtpsReliableWriterProtocol::send_window_update_period (p. 2251), **rti::core::policy::↩ RtpsReliableWriterProtocol::send_window_decrease_factor** (p. 2253)

Categories

Immutable (p. 243), **Reliability** (p. 222), **Flow Control** (p. 188)

8.322.3.45 send_window_increase_factor() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::send_window_increase_factor ( ) const
```

Getter (see setter with the same name)

8.322.3.46 send_window_decrease_factor() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::send_window_decrease_factor (
            int32_t the_send_window_decrease_factor )
```

<<***extension***>> *(p. 116)* Decreases send window size by this percentage when reacting dynamically to network conditions

The **dds::pub::DataWriter** (p. 1138)'s send window will dynamically change, between the min and max send window sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

When increased network congestion causes a negative acknowledgement to be received by a writer, the send window will be decreased by this factor to throttle the effective send rate. The factor is treated as a percentage, where a factor of 80 would decrease the send window to 80% of its previous size. The decreased send window size will not be less than the min_send_window_size.

[default] The default value depends on the container policy:

- For **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217): 70
- For **rti::core::policy::DiscoveryConfig::publication_writer** (p. 1289): 50
- For **rti::core::policy::DiscoveryConfig::subscription_writer** (p. 1291): 50
- For **rti::core::policy::DiscoveryConfig::participant_message_writer** (p. 1296): 50

[range] [0, 100]

See also

rti::core::policy::RtpsReliableWriterProtocol::send_window_update_period (p. 2251), **rti::core::policy::RtpsReliableWriterProtocol::send_window_increase_factor** (p. 2252)

Categories

Immutable (p. 243), **Reliability** (p. 222), **Flow Control** (p. 188)

8.322.3.47 send_window_decrease_factor() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::send_window_decrease_factor ( ) const
```

Getter (see setter with the same name)

8.322.3.48 enable_multicast_periodic_heartbeat() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::enable_multicast_↩
periodic_heartbeat (
            bool the_enable_multicast_periodic_heartbeat )
```

<<***extension***>> *(*p. *116)* Whether periodic heartbeat messages are sent over multicast

When enabled, if a reader has a multicast destination, then the writer will send its periodic HEARTBEAT messages to that destination. Otherwise, if not enabled or the reader does not have a multicast destination, the writer will send its periodic HEARTBEATs over unicast.

[default] false

Categories

Immutable (p. 243), **Reliability** (p. 222), **Transports** (p. 237)

8.322.3.49 enable_multicast_periodic_heartbeat() [2/2]

```
bool rti::core::policy::RtpsReliableWriterProtocol::enable_multicast_periodic_heartbeat ( ) const
```

Getter (see setter with the same name)

8.322.3.50 multicast_resend_threshold() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::multicast_resend_↩
threshold (
            int32_t the_multicast_resend_threshold )
```

<<***extension***>> *(*p. *116)* The minimum number of requesting readers needed to trigger a multicast resend

Given readers with multicast destinations, when a reader NACKs for samples to be resent, the writer can either resend them over unicast or multicast. In order for the writer to resend over multicast, this threshold is the minimum number of readers of the same multicast group that the writer must receive NACKs from within a single response-delay. This allows the writer to coalesce near-simultaneous unicast resends into a multicast resend. Note that a threshold of 1 means that all resends will be sent over multicast, if available.

[default] 2

[range] [>= 1]

Categories

Immutable (p. 243), **Reliability** (p. 222), **Transports** (p. 237)

8.322.3.51 multicast_resend_threshold() [2/2]

```
int32_t rti::core::policy::RtpsReliableWriterProtocol::multicast_resend_threshold ( ) const
```

Getter (see setter with the same name)

8.322.3.52 disable_repair_piggyback_heartbeat() [1/2]

```
RtpsReliableWriterProtocol & rti::core::policy::RtpsReliableWriterProtocol::disable_repair_↩
piggyback_heartbeat (
            bool the_disable_repair_piggyback_heartbeat )
```

<<***extension***>> *(*p. *116)* Prevents piggyback heartbeats from being sent with repair samples

When samples are repaired, the **dds::pub::DataWriter** (p. 1138) resends **rti::core::policy::RtpsReliableWriter↩ Protocol::max_bytes_per_nack_response** (p. 2244) bytes and a piggyback heartbeat with each message. You can configure the **dds::pub::DataWriter** (p. 1138) to not send the piggyback heartbeat and instead rely on the **rti::core↩ ::policy::RtpsReliableWriterProtocol::late_joiner_heartbeat_period** (p. 2236) to control the throughput used to repair samples. This field is mutable only for **rti::core::policy::DataWriterProtocol::rtps_reliable_writer** (p. 1217). **[default]** false

Categories

Immutable (p. 243), **Reliability** (p. 222), **Flow Control** (p. 188)

8.322.3.53 disable_repair_piggyback_heartbeat() [2/2]

```
bool rti::core::policy::RtpsReliableWriterProtocol::disable_repair_piggyback_heartbeat ( ) const
```

Getter (see setter with the same name)

8.323 rti::core::policy::RtpsReservedPortKindMask Class Reference

<<***extension***>> *(*p. *354)* Mask of reserved ports

```
#include <rti/core/policy/CorePolicy.hpp>
```

Inherits std::bitset< 4 >.

Public Types

- typedef std::bitset< 4 > **MaskType**

  The base type, `std::bitset`.

Public Member Functions

- **RtpsReservedPortKindMask** ()

  Creates the mask ***none()*** *(*p. *556)*
- **RtpsReservedPortKindMask** (uint64_t mask)

  Creates a mask from the bits in an integer.
- **RtpsReservedPortKindMask** (const **MaskType** &mask)

  Creates a mask from a `std::bitset`.

Static Public Member Functions

- static const **RtpsReservedPortKindMask** **all** ()

  All bits are set.
- static const **RtpsReservedPortKindMask** **none** ()

  No bits are set.
- static const **RtpsReservedPortKindMask** **default_mask** ()

  The default value of ***rti::core::policy::WireProtocol::rtps_reserved_port_mask*** *(*p. *2639).*
- static const **RtpsReservedPortKindMask** **builtin_unicast** ()

  Select the ***metatraffic*** *unicast port.*
- static const **RtpsReservedPortKindMask** **builtin_multicast** ()

  Select the ***metatraffic*** *multicast port.*
- static const **RtpsReservedPortKindMask** **user_unicast** ()

  Select the ***usertraffic*** *unicast port.*
- static const **RtpsReservedPortKindMask** **user_multicast** ()

  Select the ***usertraffic*** *multicast port.*

8.323.1 Detailed Description

<<***extension***>> *(*p. *354)* Mask of reserved ports

QoS:

rti::core::policy::WireProtocol (p. 2630)

8.323.2 Member Typedef Documentation

8.323.2.1 MaskType

```
typedef std::bitset<4>  rti::core::policy::RtpsReservedPortKindMask::MaskType
```

The base type, `std::bitset`.

8.323.3 Constructor & Destructor Documentation

8.323.3.1 RtpsReservedPortKindMask() [1/3]

```
rti::core::policy::RtpsReservedPortKindMask::RtpsReservedPortKindMask ( )  [inline]
```

Creates the mask **none()** (p. 556)

Referenced by **all()**, **builtin_multicast()**, **builtin_unicast()**, **default_mask()**, **none()**, **user_multicast()**, and **user_unicast()**.

8.323.3.2 RtpsReservedPortKindMask() [2/3]

```
rti::core::policy::RtpsReservedPortKindMask::RtpsReservedPortKindMask (
            uint64_t mask )  [inline], [explicit]
```

Creates a mask from the bits in an integer.

8.323.3.3 RtpsReservedPortKindMask() [3/3]

```
rti::core::policy::RtpsReservedPortKindMask::RtpsReservedPortKindMask (
            const  MaskType & mask )  [inline]
```

Creates a mask from a `std::bitset`.

8.323.4 Member Function Documentation

8.323.4.1 builtin_unicast()

```
static const RtpsReservedPortKindMask rti::core::policy::RtpsReservedPortKindMask::builtin_↩
unicast ( ) [inline], [static]
```

Select the **metatraffic** unicast port.

References **RtpsReservedPortKindMask()**.

8.323.4.2 builtin_multicast()

```
static const RtpsReservedPortKindMask rti::core::policy::RtpsReservedPortKindMask::builtin_↩
multicast ( ) [inline], [static]
```

Select the **metatraffic** multicast port.

References **RtpsReservedPortKindMask()**.

8.323.4.3 user_unicast()

```
static const RtpsReservedPortKindMask rti::core::policy::RtpsReservedPortKindMask::user_unicast (
) [inline], [static]
```

Select the **usertraffic** unicast port.

References **RtpsReservedPortKindMask()**.

8.323.4.4 user_multicast()

```
static const RtpsReservedPortKindMask rti::core::policy::RtpsReservedPortKindMask::user_multicast
( ) [inline], [static]
```

Select the **usertraffic** multicast port.

References **RtpsReservedPortKindMask()**.

8.324 rti::core::RtpsWellKnownPorts Class Reference

<<***extension***>> *(p. 354)* Configures the mapping of the RTPS well-known ports

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **RtpsWellKnownPorts** ()

  Creates an instance that contains the default RTPS well-known ports.
- **RtpsWellKnownPorts** (int32_t the_port_base, int32_t the_domain_id_gain, int32_t the_participant_id_gain, int32_t the_builtin_multicast_port_offset, int32_t the_builtin_unicast_port_offset, int32_t the_user_multicast_↩ port_offset, int32_t the_user_unicast_port_offset)

  Creates an instance with the specified ports.
- int32_t **port_base** () const

  Getter (see setter with the same name)
- **RtpsWellKnownPorts** & **port_base** (int32_t the_port_base)

  <<***extension***>> *(*p. *116**) The base port offset.*
- int32_t **domain_id_gain** () const

  Getter (see setter with the same name)
- **RtpsWellKnownPorts** & **domain_id_gain** (int32_t the_domain_id_gain)

  <<***extension***>> *(*p. *116**) Tunable domain gain parameter.*
- int32_t **participant_id_gain** () const

  Getter (see setter with the same name)
- **RtpsWellKnownPorts** & **participant_id_gain** (int32_t the_participant_id_gain)

  <<***extension***>> *(*p. *116**) Tunable participant gain parameter.*
- int32_t **builtin_multicast_port_offset** () const

  Getter (see setter with the same name)
- **RtpsWellKnownPorts** & **builtin_multicast_port_offset** (int32_t the_builtin_multicast_port_offset)

  <<***extension***>> *(*p. *116**) Additional offset for* ***metatraffic*** *multicast port.*
- int32_t **builtin_unicast_port_offset** () const

  Getter (see setter with the same name)
- **RtpsWellKnownPorts** & **builtin_unicast_port_offset** (int32_t the_builtin_unicast_port_offset)

  <<***extension***>> *(*p. *116**) Additional offset for* ***metatraffic*** *unicast port.*
- int32_t **user_multicast_port_offset** () const

  Getter (see setter with the same name)
- **RtpsWellKnownPorts** & **user_multicast_port_offset** (int32_t the_user_multicast_port_offset)

  <<***extension***>> *(*p. *116**) Additional offset for* ***usertraffic*** *multicast port.*
- int32_t **user_unicast_port_offset** () const

  Getter (see setter with the same name)
- **RtpsWellKnownPorts** & **user_unicast_port_offset** (int32_t the_user_unicast_port_offset)

  <<***extension***>> *(*p. *116**) Additional offset for* ***usertraffic*** *unicast port.*

Static Public Member Functions

- static **RtpsWellKnownPorts Interoperable** ()

  Returns an instance containing the port mapping compliant with the OMG DDS Interoperability wire protocol.
- static **RtpsWellKnownPorts BackwardsCompatible** ()

  Returns an instance containing the port mapping compatible with previous versions of RTI Connext.

8.324.1 Detailed Description

<<***extension***>> *(p. 354)* Configures the mapping of the RTPS well-known ports

RTI Connext uses the RTPS wire protocol. The discovery protocols defined by RTPS rely on well-known ports to initiate discovery. These well-known ports define the multicast and unicast ports on which a Participant will listen for discovery **metatraffic** from other Participants. The discovery metatraffic contains all the information required to establish the presence of remote DDS entities in the network.

The well-known ports are defined by RTPS in terms of port mapping expressions with several tunable parameters, which allow you to customize what network ports are used by RTI Connext. These parameters are exposed in **rti::core::RtpsWellKnownPorts** (p. 2258). In order for all Participants in a system to correctly discover each other, it is important that they all use the same port mapping expressions.

The actual port mapping expressions, as defined by the RTPS specification, can be found below. In addition to the parameters listed in **rti::core::RtpsWellKnownPorts** (p. 2258), the port numbers depend on:

- `domain_id`, as specified in **dds::domain::DomainParticipant()** (p. 1326)
- `participant_id`, as specified using **rti::core::policy::WireProtocol::participant_id** (p. 2635)

The `domain_id` parameter ensures no port conflicts exist between Participants belonging to different domains. This also means that discovery metatraffic in one domain is not visible to Participants in a different domain. The `participant_id` parameter ensures that unique unicast port numbers are assigned to Participants belonging to the same domain on a given host.

The *metatraffic_unicast_port* is used to exchange discovery metatraffic using unicast.

```
metatraffic_unicast_port = port_base + (domain_id_gain * domain_id) + (participant_id_gain * participant_id) + b
```

The *metatraffic_multicast_port* is used to exchange discovery metatraffic using multicast. The corresponding multicast group addresses are specified via **rti::core::policy::Discovery::multicast_receive_addresses** (p. 1268) on a **dds::domain::DomainParticipant** (p. 1326) entity.

```
metatraffic_multicast_port = port_base + (domain_id_gain * domain_id) + builtin_multicast_port_offset
```

RTPS also defines the *default* multicast and unicast ports on which DataReaders and DataWriters receive **usertraffic**. These default ports can be overridden using the **rti::core::policy::TransportMulticast** (p. 2543), **rti::core::policy::TransportUnicast** (p. 2556), or by the **rti::core::policy::TransportUnicast** (p. 2556) QoS policies.

The *usertraffic_unicast_port* is used to exchange user data using unicast.

```
usertraffic_unicast_port = port_base + (domain_id_gain * domain_id) + (participant_id_gain * participant_id) + u
```

The *usertraffic_multicast_port* is used to exchange user data using multicast. The corresponding multicast group addresses can be configured using **rti::core::policy::TransportMulticast** (p. 2543).

```
usertraffic_multicast_port = port_base + (domain_id_gain * domain_id) + user_multicast_port_offset
```

By default, the port mapping parameters are configured to compliant with OMG's DDS Interoperability Wire Protocol (see also **rti::core::RtpsWellKnownPorts::Interoperable()** (p. 557)).

The OMG's DDS Interoperability Wire Protocol compliant port mapping parameters are *not* backwards compatible with previous versions of the RTI Connext middleware.

When modifying the port mapping parameters, care must be taken to avoid port aliasing. This would result in undefined discovery behavior. The chosen parameter values will also determine the maximum possible number of domains in the system and the maximum number of participants per domain. Additionally, any resulting mapped port number must be within the range imposed by the underlying transport. For example, for UDPv4, this range typically equals [1024 - 65535].

Note: On Windows, you should avoid using ports 49152 through 65535 for inbound traffic. RTI Connext's ephemeral ports (see "Ports Used for Communication" in the `User's Manual`) may be within that range (see `https://msdn.microsoft.com/en-us/library/windows/desktop/ms737550(v=vs.85).aspx`). With the default **RtpsWellKnownPorts** (p. 2258) settings, port 49152 corresponds to domain ID 167, so using domain IDs 168 through 232 on Windows introduces the risk of a port collision and failure to create the Domain Participant when using multicast discovery. You may see this error:

RTIOsapiSocket_bindWithIP:OS bind() failure, error 0X271D: An attempt was made to access a socket in a way forbidden by its access permissions.

QoS:

rti::core::policy::WireProtocol (p. 2630)

8.324.2 Constructor & Destructor Documentation

8.324.2.1 RtpsWellKnownPorts() [1/2]

```
rti::core::RtpsWellKnownPorts::RtpsWellKnownPorts ( )  [inline]
```

Creates an instance that contains the default RTPS well-known ports.

8.324.2.2 RtpsWellKnownPorts() [2/2]

```
rti::core::RtpsWellKnownPorts::RtpsWellKnownPorts (
            int32_t the_port_base,
            int32_t the_domain_id_gain,
            int32_t the_participant_id_gain,
            int32_t the_builtin_multicast_port_offset,
            int32_t the_builtin_unicast_port_offset,
            int32_t the_user_multicast_port_offset,
            int32_t the_user_unicast_port_offset )  [inline]
```

Creates an instance with the specified ports.

See individual setters.

8.324.3 Member Function Documentation

8.324.3.1 port_base() [1/2]

```
int32_t rti::core::RtpsWellKnownPorts::port_base ( ) const  [inline]
```

Getter (see setter with the same name)

8.324.3.2 port_base() [2/2]

```
RtpsWellKnownPorts & rti::core::RtpsWellKnownPorts::port_base (
            int32_t the_port_base )  [inline]
```

<<***extension***>> *(*p. *116)* The base port offset.

All mapped well-known ports are offset by this value.

[default] 7400

[range] [>= 1], but resulting ports must be within the range imposed by the underlying transport.

8.324.3.3 domain_id_gain() [1/2]

```
int32_t rti::core::RtpsWellKnownPorts::domain_id_gain ( ) const  [inline]
```

Getter (see setter with the same name)

8.324.3.4 domain_id_gain() [2/2]

```
RtpsWellKnownPorts & rti::core::RtpsWellKnownPorts::domain_id_gain (
            int32_t the_domain_id_gain )  [inline]
```

<<***extension***>> *(*p. *116)* Tunable domain gain parameter.

Multiplier of the `domain_id`. Together with `participant_id_gain`, it determines the highest `domain_id` and `participant_id` allowed on this network.

In general, there are two ways to setup `domain_id_gain` and `participant_id_gain` parameters.

If `domain_id_gain` > `participant_id_gain`, it results in a port mapping layout where all **dds::domain::DomainParticipant** (p. 1326) instances within a single domain occupy a consecutive range of `domain_id_gain` ports. Precisely, all ports occupied by the domain fall within:

```
(port_base + (domain_id_gain * domain_id))
```

and:

```
(port_base + (domain_id_gain * (domain_id + 1)) - 1)
```

Under such a case, the highest `domain_id` is limited only by the underlying transport's maximum port. The highest `participant_id`, however, must satisfy:

```
max_participant_id < (domain_id_gain / participant_id_gain)
```

On the contrary, if `domain_id_gain` <= `participant_id_gain`, it results in a port mapping layout where a given domain's **dds::domain::DomainParticipant** (p. 1326) instances occupy ports spanned across the entire valid port range allowed by the underlying transport. For instance, it results in the following potential mapping:

Mapped Port	Domain Id	Participant ID
higher port number	Domain Id = 1	Participant ID = 2
	Domain Id = 0	Participant ID = 2
	Domain Id = 1	Participant ID = 1
	Domain Id = 0	Participant ID = 1
	Domain Id = 1	Participant ID = 0
lower port number	Domain Id = 0	Participant ID = 0

Under this case, the highest `participant_id` is limited only by the underlying transport's maximum port. The highest `domain_id`, however, must satisfy:

```
max_domain_id < (participant_id_gain / domain_id_gain)
```

Additionally, `domain_id_gain` also determines the range of the port-specific offsets.

```
domain_id_gain > abs(builtin_multicast_port_offset - user_multicast_port_offset)
```

```
domain_id_gain > abs(builtin_unicast_port_offset - user_unicast_port_offset)
```

Violating this may result in port aliasing and undefined discovery behavior.

[default] 250

[range] [$>$ 0], but resulting ports must be within the range imposed by the underlying transport.

8.324.3.5 participant_id_gain() [1/2]

```
int32_t rti::core::RtpsWellKnownPorts::participant_id_gain ( ) const  [inline]
```

Getter (see setter with the same name)

8.324.3.6 participant_id_gain() [2/2]

```
RtpsWellKnownPorts & rti::core::RtpsWellKnownPorts::participant_id_gain (
            int32_t the_participant_id_gain )  [inline]
```

<<***extension***>> *(p. 116)* Tunable participant gain parameter.

Multiplier of the `participant_id`. See **rti::core::RtpsWellKnownPorts::domain_id_gain** (p. 2262) for its implications on the highest `domain_id` and `participant_id` allowed on this network.

Additionally, `participant_id_gain` also determines the range of `builtin_unicast_port_offset` and `user_unicast_port_offset`.

```
participant_id_gain > abs(builtin_unicast_port_offset - user_unicast_port_offset)
```

[default] 2

[range] [$>$ 0], but resulting ports must be within the range imposed by the underlying transport.

8.324.3.7 builtin_multicast_port_offset() [1/2]

```
int32_t rti::core::RtpsWellKnownPorts::builtin_multicast_port_offset ( ) const  [inline]
```

Getter (see setter with the same name)

8.324.3.8 builtin_multicast_port_offset() [2/2]

```
RtpsWellKnownPorts & rti::core::RtpsWellKnownPorts::builtin_multicast_port_offset (
            int32_t the_builtin_multicast_port_offset )  [inline]
```

<<***extension***>> *(*p. *116)* Additional offset for **metatraffic** multicast port.

It must be unique from other port-specific offsets.

[default] 0

[range] [>= 0], but resulting ports must be within the range imposed by the underlying transport.

8.324.3.9 builtin_unicast_port_offset() [1/2]

```
int32_t rti::core::RtpsWellKnownPorts::builtin_unicast_port_offset ( ) const  [inline]
```

Getter (see setter with the same name)

8.324.3.10 builtin_unicast_port_offset() [2/2]

```
RtpsWellKnownPorts & rti::core::RtpsWellKnownPorts::builtin_unicast_port_offset (
            int32_t the_builtin_unicast_port_offset )  [inline]
```

<<***extension***>> *(*p. *116)* Additional offset for **metatraffic** unicast port.

It must be unique from other port-specific offsets.

[default] 10

[range] [>= 0], but resulting ports must be within the range imposed by the underlying transport.

8.324.3.11 user_multicast_port_offset() [1/2]

```
int32_t rti::core::RtpsWellKnownPorts::user_multicast_port_offset ( ) const  [inline]
```

Getter (see setter with the same name)

8.324.3.12 user_multicast_port_offset() [2/2]

```
RtpsWellKnownPorts & rti::core::RtpsWellKnownPorts::user_multicast_port_offset (
            int32_t the_user_multicast_port_offset )  [inline]
```

<<***extension***>> *(*p. *116)* Additional offset for **usertraffic** multicast port.

It must be unique from other port-specific offsets.

[default] 1

[range] [>= 0], but resulting ports must be within the range imposed by the underlying transport.

8.324.3.13 user_unicast_port_offset() [1/2]

```
int32_t rti::core::RtpsWellKnownPorts::user_unicast_port_offset ( ) const  [inline]
```

Getter (see setter with the same name)

8.324.3.14 user_unicast_port_offset() [2/2]

```
RtpsWellKnownPorts & rti::core::RtpsWellKnownPorts::user_unicast_port_offset (
            int32_t the_user_unicast_port_offset )  [inline]
```

<<***extension***>> *(*p. *116)* Additional offset for **usertraffic** unicast port.

It must be unique from other port-specific offsets.

[default] 11

[range] [>= 0], but resulting ports must be within the range imposed by the underlying transport.

8.325 dds::core::safe_enum< def, inner > Class Template Reference

<<***value-type***>> *(*p. *350)* Provides a safe, scoped enumeration based on def::type

```
#include <SafeEnumeration.hpp>
```

Inherits def.

Public Types

- typedef inner **inner_enum**

Public Member Functions

- **safe_enum** ()
  Initializes the enumeration to zero.
- **safe_enum** (**inner_enum** v)
  Copy constructor.
- **inner_enum underlying** () const
  Retrieves the actual `enum` *value.*
- bool **operator==** (const **safe_enum** &s) const
  Applies operator to underlying enumeration value.
- bool **operator!=** (const **safe_enum** &s) const
  Applies operator to underlying enumeration value.
- bool **operator**< (const **safe_enum** &s) const
  Applies operator to underlying enumeration value.
- bool **operator**<**=** (const **safe_enum** &s) const
  Applies operator to underlying enumeration value.
- bool **operator**> (const **safe_enum** &s) const
  Applies operator to underlying enumeration value.
- bool **operator**>**=** (const **safe_enum** &s) const
  Applies operator to underlying enumeration value.

Friends

- void **swap** (**safe_enum** &left, **safe_enum** &right) OMG_NOEXCEPT
  Swaps the enumeration values.

Related Functions

(Note that these are not member functions.)

- template<typename def , typename inner >
  std::ostream & **operator**<< (std::ostream &out, const **safe_enum**< def, inner > &the_enum)
  Applies `operator<<` *to the underlying enum.*

8.325.1 Detailed Description

template<typename def, typename inner = typename def::type>
class dds::core::safe_enum< def, inner >

<<***value-type***>> *(*p. *350)* Provides a safe, scoped enumeration based on def::type

The purpose of this class is to provide a C++11-style enum class, where the enumeration constants are scoped.

Template Parameters

def	A struct containing the actual enumeration, `def::type`

See for example **dds::core::policy::ReliabilityKind** (p. 545). The actual enumeration is in **dds::core::policy::ReliabilityKind_def** (p. 2156), but the application only needs to use the former:

```
using dds::core::policy::ReliabilityKind;
ReliabilityKind kind = ReliabilityKind::RELIABLE;
```

A **safe_enum** (p. 2265) can be converted to and from an integer.

To convert to an integer, use the member function **underlying()** (p. 2268):

```
ReliabilityKind reliability = ReliabilityKind::RELIABLE;
int reliability_as_int = reliability.underlying();
```

To convert from an integer, `static_cast` it to the member type `inner_enum`:

```
int reliability_as_int = 2;
ReliabilityKind reliability = static_cast<ReliabilityKind::inner_enum>(reliability_as_int);
```

8.325.2 Member Typedef Documentation

8.325.2.1 inner_enum

```
template<typename def , typename inner = typename def::type>
typedef inner  dds::core::safe_enum< def, inner >::inner_enum
```

The underlying `enum` type

To convert from an integer, `static_cast` it to the member type `inner_enum`:

```
int reliability_as_int = 2;
ReliabilityKind reliability = static_cast<ReliabilityKind::inner_enum>(reliability_as_int);
```

8.325.3 Constructor & Destructor Documentation

8.325.3.1 safe_enum() [1/2]

```
template<typename def , typename inner = typename def::type>
dds::core::safe_enum< def, inner >::safe_enum ( )  [inline]
```

Initializes the enumeration to zero.

Note

Zero may not be a valid enumerator. It's best to assign a valid enumerator whenever possible:

```
Reliability reliability = Reliability::RELIABLE;
```

8.325.3.2 safe_enum() [2/2]

```
template<typename def , typename inner = typename def::type>
dds::core::safe_enum< def, inner >::safe_enum (
              inner_enum v )  [inline]
```

Copy constructor.

8.325.4 Member Function Documentation

8.325.4.1 underlying()

```
template<typename def , typename inner = typename def::type>
inner_enum  dds::core::safe_enum< def, inner >::underlying ( ) const  [inline]
```

Retrieves the actual enum value.

Referenced by **dds::core::safe_enum< def, inner >::operator<<()**.

8.325.4.2 operator==()

```
template<typename def , typename inner = typename def::type>
bool  dds::core::safe_enum< def, inner >::operator== (
             const  safe_enum< def, inner > & s ) const  [inline]
```

Applies operator to underlying enumeration value.

8.325.4.3 operator"!=()

```
template<typename def , typename inner = typename def::type>
bool  dds::core::safe_enum< def, inner >::operator!= (
             const  safe_enum< def, inner > & s ) const  [inline]
```

Applies operator to underlying enumeration value.

8.325.4.4 operator<()

```
template<typename def , typename inner = typename def::type>
bool  dds::core::safe_enum< def, inner >::operator< (
            const  safe_enum< def, inner > & s ) const  [inline]
```

Applies operator to underlying enumeration value.

8.325.4.5 operator<=()

```
template<typename def , typename inner = typename def::type>
bool  dds::core::safe_enum< def, inner >::operator<= (
            const  safe_enum< def, inner > & s ) const  [inline]
```

Applies operator to underlying enumeration value.

8.325.4.6 operator>()

```
template<typename def , typename inner = typename def::type>
bool  dds::core::safe_enum< def, inner >::operator> (
            const  safe_enum< def, inner > & s ) const  [inline]
```

Applies operator to underlying enumeration value.

8.325.4.7 operator>=()

```
template<typename def , typename inner = typename def::type>
bool  dds::core::safe_enum< def, inner >::operator>= (
            const  safe_enum< def, inner > & s ) const  [inline]
```

Applies operator to underlying enumeration value.

8.325.5 Friends And Related Function Documentation

8.325.5.1 swap

```
template<typename def , typename inner = typename def::type>
void swap (
              safe_enum< def, inner > & left,
              safe_enum< def, inner > & right )  [friend]
```

Swaps the enumeration values.

8.325.5.2 operator<<()

```
template<typename def , typename inner >
std::ostream & operator<< (
             std::ostream & out,
             const  safe_enum< def, inner > & the_enum )  [related]
```

Applies operator<< to the underlying enum.

References **dds::core::safe_enum< def, inner >::underlying()**.

8.326 dds::sub::Sample< T > Class Template Reference

<<***value-type***>> *(*p. *350)* This class encapsulate the data and meta-data associated with DDS samples.

```
#include "dds/sub/Sample.hpp"
```

Public Member Functions

- **Sample** ()=default
  Create a sample.
- **Sample** (const T &the_data, const **SampleInfo** &the_info)
  *Create a **Sample** (*p. *2270) with the provided data and **SampleInfo** (*p. *2285).*
- **Sample** (const **rti::sub::LoanedSample**< T > &loaned_sample)
  <<***extension***>> *(*p. *354) Construct a **Sample** (*p. *2270) from a LoanedSample*
- **Sample** (const **Sample** &other)=default
  Copy constructor.
- **Sample** & **operator=** (const **Sample** &other)=default
  Copy assignment operator.
- const DataType & **data** () const
  *Get the data of type T associated with this **Sample** (*p. *2270).*
- void **data** (const DataType &the_data)
  *Set the data of type T associated with this **Sample** (*p. *2270).*
- const **SampleInfo** & **info** () const
  *Get the **SampleInfo** (*p. *2285) associated with this **Sample** (*p. *2270).*
- void **info** (const **SampleInfo** &the_info)
  *Set the **SampleInfo** (*p. *2285) associated with this **Sample** (*p. *2270).*
- **Sample** & **operator=** (const **rti::sub::LoanedSample**< T > &loaned_sample)
  <<***extension***>> *(*p. *354) Assignment operator from an **rti::sub::LoanedSample** (*p. *1659)*

8.326.1 Detailed Description

template<typename T>
class dds::sub::Sample< T >

<<***value-type***>> *(p. 350)* This class encapsulate the data and meta-data associated with DDS samples.

Template Parameters

The	type of the data that **Sample** (p. 2270) represents

8.326.2 Constructor & Destructor Documentation

8.326.2.1 Sample() [1/4]

```
template<typename T >
dds::sub::Sample< T >::Sample ( )  [default]
```

Create a sample.

Postcondition

The value of **data()** (p. 2272) is default-constructed, and **info()** (p. 2273).valid() is false.

8.326.2.2 Sample() [2/4]

```
template<typename T >
dds::sub::Sample< T >::Sample (
            const T & the_data,
            const  SampleInfo & the_info )  [inline]
```

Create a **Sample** (p. 2270) with the provided data and **SampleInfo** (p. 2285).

Parameters

the_data	The data to create the **Sample** (p. 2270) with
the_info	The **SampleInfo** (p. 2285) to create the **Sample** (p. 2270) with

8.326.2.3 Sample() [3/4]

```
template<typename T >
dds::sub::Sample< T >::Sample (
            const  rti::sub::LoanedSample< T > & loaned_sample )  [inline], [explicit]
```

<<***extension***>> *(p. 354)* Construct a **Sample** (p. 2270) from a LoanedSample

If `loaned_sample.info()`.valid() is false, this sample's data value is default-constructed.

8.326.2.4 Sample() [4/4]

```
template<typename T >
dds::sub::Sample< T >::Sample (
            const  Sample< T > & other )  [default]
```

Copy constructor.

8.326.3 Member Function Documentation

8.326.3.1 operator=() [1/2]

```
template<typename T >
Sample &  dds::sub::Sample< T >::operator= (
            const  Sample< T > & other )  [default]
```

Copy assignment operator.

8.326.3.2 data() [1/2]

```
template<typename T >
const DataType &  dds::sub::Sample< T >::data ( ) const  [inline]
```

Get the data of type T associated with this **Sample** (p. 2270).

This function returns the current value of the data even if **info()** (p. 2273).valid() is false.

8.326.3.3 data() [2/2]

```
template<typename T >
void  dds::sub::Sample< T >::data (
            const DataType & the_data )  [inline]
```

Set the data of type T associated with this **Sample** (p. 2270).

8.326.3.4 info() [1/2]

```
template<typename T >
const  SampleInfo &  dds::sub::Sample< T >::info ( ) const  [inline]
```

Get the **SampleInfo** (p. 2285) associated with this **Sample** (p. 2270).

Referenced by **rti::request::IsReplyRelatedPredicate< T >::operator()()**.

8.326.3.5 info() [2/2]

```
template<typename T >
void  dds::sub::Sample< T >::info (
            const  SampleInfo & the_info )  [inline]
```

Set the **SampleInfo** (p. 2285) associated with this **Sample** (p. 2270).

8.326.3.6 operator=() [2/2]

```
template<typename T >
Sample &  dds::sub::Sample< T >::operator= (
            const  rti::sub::LoanedSample< T > & loaned_sample )  [inline]
```

<<***extension***>> *(p. 354)* Assignment operator from an **rti::sub::LoanedSample** (p. 1659)

This operator allows obtaining a copy of both the data and the info from a LoanedSample.

Postcondition

If `loaned_sample.info()`.valid() is false, this sample's **data()** (p. 2272) value is not modified, but **info()** (p. 2273).valid() is false.

8.327 rti::flat::Sample< OffsetType > Class Template Reference

The generic definition of FlatData topic-types.

`#include <FlatSample.hpp>`

Inherits rti::flat::SampleBase.

Public Types

- typedef OffsetType **Offset**
  The related Offset type.
- typedef OffsetType::ConstOffset **ConstOffset**
  The related read-only Offset type.

Public Member Functions

- **Offset root** ()
  Provides the Offset to the top-level type.
- **ConstOffset root** () const
  Provides the Offset to the top-level type.
- **Sample**< OffsetType > * **clone** () const
  *Clones a **Sample** (p. 2274), creating an unmanaged **Sample** (p. 2274).*

Static Public Member Functions

- static **Sample**< OffsetType > * **create_data** ()
  *Creates an unmanaged FlatData **Sample** (p. 2274).*
- static void **delete_data** (**rti::flat::Sample**< OffsetType > *sample)
  *Releases an unmanaged **Sample** (p. 2274).*

8.327.1 Detailed Description

template<typename OffsetType>
class rti::flat::Sample< OffsetType >

The generic definition of FlatData topic-types.

Template Parameters

OffsetType	The Offset to the beginning of the data **Sample** (p. 2274) (the root), for example **MyFlatFinalOffset** (p. 1759), or **MyFlatMutableOffset** (p. 1770).

For a FlatData IDL type, **rtiddsgen** generates an instantiation of this class, such as MyFlatFinal or MyFlatMutable. That's the **topic-type** used to declare a **dds::topic::Topic** (p. 2474) and write or read FlatData samples.

A FlatData **Sample** (p. 2274) owns an inline buffer that contains the serialized data. To access the **Sample** (p. 2274) data members we use **Offsets** (p. 414), iterators that point to the position of a member in the **Sample** (p. 2274) buffer. The **root()** (p. 2276) is the Offset to the top-level element (for example **MyFlatMutableOffset** (p. 1770)). From there **MyFlatMutableOffset** (p. 1770)'s member functions provide access to its members.

The method to create a **Sample** (p. 2274) varies depending on whether the type is final or mutable.

Samples of **final** FlatData types always have the same size, and can be obtained directly with **dds::pub::DataWriter::get_loan()** (p. 1183).

Samples of **mutable** FlatData types need to be "built" using the Builder returned by **rti::flat::build_data()** (p. 411) (**MyFlatMutableBuilder** (p. 1763) in this example). A Builder allows adding each member individually in any order, and sizing sequences as needed.

See **Publishing FlatData** (p. 420) for code snippets.

Samples created with **dds::pub::DataWriter::get_loan()** (p. 1183) or **rti::flat::build_data()** (p. 411) are DataWriter-managed and do not need to be explicitly deleted. See **dds::pub::DataWriter::get_loan()** (p. 1183) for an explanation of the lifecycle of DataWriter-managed samples.

Samples received with a **dds::sub::DataReader** (p. 976) can be examined by obtaining the **root()** (p. 2276), but cannot be modified. See **Subscribing to FlatData** (p. 421).

FlatData samples are not value types. They don't provide copy constructors or assignment operators, because of their definition as an inline buffer that contains the serialized **Sample** (p. 2274) at all times. A **Sample** (p. 2274) can be cloned with the static function **clone()** (p. 2277). A cloned **Sample** (p. 2274) is unmanaged and has to be deleted with the static function **delete_data()** (p. 2277).

8.327.2 Member Typedef Documentation

8.327.2.1 Offset

```
template<typename OffsetType >
typedef OffsetType  rti::flat::Sample< OffsetType >::Offset
```

The related Offset type.

This is the template parameter, OffsetType

8.327.2.2 ConstOffset

```
template<typename OffsetType >
typedef OffsetType::ConstOffset  rti::flat::Sample< OffsetType >::ConstOffset
```

The related read-only Offset type.

For example, **MyFlatMutableOffset::ConstOffset** (p. 1771).

8.327.3 Member Function Documentation

8.327.3.1 root() [1/2]

```
template<typename OffsetType >
Offset rti::flat::Sample< OffsetType >::root ( ) [inline]
```

Provides the Offset to the top-level type.

Returns

An Offset that allows reading and modifying the members of this **Sample** (p. 2274), for example **MyFlatFinalOffset** (p. 1759) or **MyFlatMutableOffset** (p. 1770).

This overload allows modifying the **Sample** (p. 2274).

Example:

```
MyFlatFinal *my_sample = ...; // for example, created with DataWriter::get_loan()
MyFlatFinalOffset my_sample_root = my_sample->root();
my_sample_root.my_primitive(33);
auto my_member_offset = my_sample_root.my_complex();
// ... modify my_member_offset
```

8.327.3.2 root() [2/2]

```
template<typename OffsetType >
ConstOffset rti::flat::Sample< OffsetType >::root ( ) const [inline]
```

Provides the Offset to the top-level type.

Returns

An Offset that allows reading the members of this **Sample** (p. 2274). For example, **MyFlatFinalOffset::ConstOffset** (p. 1760) or **MyFlatMutableOffset::ConstOffset** (p. 1771).

This overload doesn't allow modifying the **Sample** (p. 2274).

Example:

```
const MyFlatMutable *my_sample = ...; // for example, read from a DataReader
MyFlatMutableOffset::ConstOffset my_sample_root = my_sample->root();
std::cout « my_sample_root.my_primitive() « std::endl;
auto my_member_offset = my_sample_root.my_mutable_seq();
// ... read my_member_offset
```

8.327.3.3 create_data()

```
template<typename OffsetType >
Sample< OffsetType > * rti::flat::Sample< OffsetType >::create_data [static]
```

Creates an unmanaged FlatData **Sample** (p. 2274).

Note

In general on the publication side it is recommended to create DataWriter-managed samples using **dds::pub::DataWriter::get_loan** (p. 1183) (for final types) and **rti::flat::build_data()** (p. 411) (for mutable types).

If the topic-type is final, the returned **Sample** (p. 2274) can be initialized by obtaining the **root()** (p. 2276).

If the topic-type is mutable, the resulting **Sample** (p. 2274) has no members, and **root()** (p. 2276) returns a **null** (p. 1876) Offset. The result of **create_data()** (p. 2276) must be used to create a Builder. This is a rare use case, and **rti::flat::build_data()** (p. 411) should be used in most situations.

In all cases, the returned **Sample** (p. 2274) must be explicitly deleted with **delete_data()** (p. 2277).

8.327.3.4 clone()

```
template<typename OffsetType >
Sample< OffsetType > * rti::flat::Sample< OffsetType >::clone ( ) const [inline]
```

Clones a **Sample** (p. 2274), creating an unmanaged **Sample** (p. 2274).

Warning

When this **Sample** (p. 2274) has been created with **dds::pub::DataWriter::get_loan** (p. 1183) or **rti::flat::build_data()** (p. 411), the clone this function creates cannot be used in **dds::pub::DataWriter::write()** (p. 1148), because the DataWriter is managing the lifecycle of its samples.

Creates a new **Sample** (p. 2274) and copies the underlying serialized buffer.

delete_data() (p. 2277) must be called to release the **Sample** (p. 2274).

8.327.3.5 delete_data()

```
template<typename OffsetType >
static void rti::flat::Sample< OffsetType >::delete_data (
            rti::flat::Sample< OffsetType > * sample ) [inline], [static]
```

Releases an unmanaged **Sample** (p. 2274).

Unmanaged samples are those created with **create_data()** (p. 2276) or **clone()** (p. 2277).

Precondition

`sample` cannot have been created with **dds::pub::DataWriter::get_loan** (p. 1183) or **rti::flat::build_data()** (p. 411).

Postcondition

`sample` becomes invalid and shouldn't be used

8.328 rti::core::SampleFlag Class Reference

<<***extension***>> *(*p. *354)* A set of flags that can be associated with a sample.

```
#include "rti/core/SampleFlag.hpp"
```

Inherits std::bitset< 32 >.

Public Types

- typedef std::bitset< 32 > **MaskType**

  A typedef of std::bitset<32> for convenience.

Public Member Functions

- **SampleFlag** ()

  Construct an empty ***SampleFlag*** *(*p. *2278) with no bits set.*
- **SampleFlag** (uint64_t mask)

  Construct a ***SampleFlag*** *(*p. *2278) from an integer.*
- **SampleFlag** (const **MaskType** &mask)

  Construct a ***SampleFlag*** *(*p. *2278) from a MaskType object.*

Static Public Member Functions

- static const **SampleFlag redelivered** ()

  Indicates that a sample has been redelivered by RTI Queuing Service.
- static const **SampleFlag intermediate_reply_sequence** ()

  Indicates that a response sample is not the last response sample for a given request. This bit is usually set by Connext Repliers sending multiple responses for a request.
- static const **SampleFlag replicate** ()

  Indicates if a sample must be broadcast by one RTI Queuing Service replica to other replicas.
- static const **SampleFlag last_shared_reader_queue** ()

  Indicates that a sample is the last sample in a SharedReaderQueue for a QueueConsumer DataReader.
- static const **SampleFlag intermediate_topic_query_sample** ()

  Indicates that a sample for a TopicQuery will be followed by more samples.
- static const **SampleFlag incomplete_snapshot_topic_query** ()

  Indicates that some of the samples that were published as part of a snapshot TopicQuery were lost. This flag is set in the last sample of the snapshot.
- static const **SampleFlag writer_removed_batch_sample** ()

  This flag will be set if a sample was accepted into the DataReader queue even though it was marked by the DataWriter as removed.
- static const **SampleFlag discovery_service_sample** ()

  This flag will be set if the sample was sent by Cloud Discovery Service.

8.328.1 Detailed Description

<<***extension***>> *(p. 354)* A set of flags that can be associated with a sample.

A set of flags that can be associated with a sample.

- Least-significant bits [0-7] are reserved by RTI
- Least-significant bits [8-15] are application specific
- Least-significant bits [16-31] are invalid and cannot be used

8.328.2 Member Typedef Documentation

8.328.2.1 MaskType

```
typedef std::bitset<32>  rti::core::SampleFlag::MaskType
```

A typedef of std::bitset<32> for convenience.

8.328.3 Constructor & Destructor Documentation

8.328.3.1 SampleFlag() [1/3]

```
rti::core::SampleFlag::SampleFlag ( )  [inline]
```

Construct an empty **SampleFlag** (p. 2278) with no bits set.

8.328.3.2 SampleFlag() [2/3]

```
rti::core::SampleFlag::SampleFlag (
            uint64_t mask )  [inline], [explicit]
```

Construct a **SampleFlag** (p. 2278) from an integer.

Parameters

mask	Value whose bits are copied to the bitset positions

8.328.3.3 SampleFlag() [3/3]

```
rti::core::SampleFlag::SampleFlag (
            const  MaskType & mask )  [inline]
```

Construct a **SampleFlag** (p. 2278) from a MaskType object.

Parameters

mask	A std::bitset<32> to construct this **SampleFlag** (p. 2278) from

8.328.4 Member Function Documentation

8.328.4.1 redelivered()

```
static const  SampleFlag rti::core::SampleFlag::redelivered ( )  [inline], [static]
```

Indicates that a sample has been redelivered by RTI Queuing Service.

8.328.4.2 intermediate_reply_sequence()

```
static const  SampleFlag rti::core::SampleFlag::intermediate_reply_sequence ( )  [inline], [static]
```

Indicates that a response sample is not the last response sample for a given request. This bit is usually set by Connext Repliers sending multiple responses for a request.

8.328.4.3 replicate()

```
static const  SampleFlag rti::core::SampleFlag::replicate ( )  [inline], [static]
```

Indicates if a sample must be broadcast by one RTI Queuing Service replica to other replicas.

8.328.4.4 last_shared_reader_queue()

```
static const SampleFlag rti::core::SampleFlag::last_shared_reader_queue ( ) [inline], [static]
```

Indicates that a sample is the last sample in a SharedReaderQueue for a QueueConsumer DataReader.

8.328.4.5 intermediate_topic_query_sample()

```
static const SampleFlag rti::core::SampleFlag::intermediate_topic_query_sample ( ) [inline],
[static]
```

Indicates that a sample for a TopicQuery will be followed by more samples.

This flag only applies to samples that have been published as a response to a **rti::sub::TopicQuery** (p. 2516).

When this bit is not set and **dds::sub::SampleInfo::topic_query_guid** (p. 2294) is different from **rti::core::Guid↩ ::unknown()** (p. 1597), this sample is the last sample for that TopicQuery coming from the DataWriter identified by **dds::sub::SampleInfo::source_guid** (p. 2293).

8.328.4.6 incomplete_snapshot_topic_query()

```
static const SampleFlag rti::core::SampleFlag::incomplete_snapshot_topic_query ( ) [inline],
[static]
```

Indicates that some of the samples that were published as part of a snapshot TopicQuery were lost. This flag is set in the last sample of the snapshot.

This flag is set in the last sample of the snapshot.

8.328.4.7 writer_removed_batch_sample()

```
static const SampleFlag rti::core::SampleFlag::writer_removed_batch_sample ( ) [inline], [static]
```

This flag will be set if a sample was accepted into the DataReader queue even though it was marked by the DataWriter as removed.

A sample can be marked as removed by the DataWriter in a batch when it is replaced due to the KEEP_LAST_↩ HISTORY_QOS QoS or because the duration in LifespanQosPolicy was reached.

If the DataReader sets the property "dds.data_reader.accept_writer_removed_batch_samples" to true, the removed sample will be accepted into the DataReader queue and this flag will be set.

A sample can be marked as removed by the DataWriter in a batch when it is replaced due to the **dds::core::policy::↩ HistoryKind_def::KEEP_LAST** (p. 1607) **dds::core::policy::History** (p. 1602) QoS or because the duration in **dds↩ ::core::policy::Lifespan** (p. 1635) was reached.

If the DataReader sets the property "dds.data_reader.accept_writer_removed_batch_samples" to true, the removed sample will be accepted into the DataReader queue and this flag will be set.

8.328.4.8 discovery_service_sample()

```
static const  SampleFlag rti::core::SampleFlag::discovery_service_sample ( )  [inline], [static]
```

This flag will be set if the sample was sent by Cloud Discovery Service.

The samples sent by Cloud Discovery Service are participant announcement samples on the ParticipantBuiltinTopic.

8.329 rti::core::SampleIdentity Class Reference

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* A **SampleIdentity** (p. 2282) defines a pair (Virtual Writer **Guid** (p. 1596), **SequenceNumber** (p. 2335)) that uniquely identifies a sample within a DDS domain and a Topic.

```
#include "rti/core/SampleIdentity.hpp"
```

Public Member Functions

- **SampleIdentity** ()
  *Create a default **SampleIdentity** (p. 2282) object.*
- **SampleIdentity** (const **Guid** &the_writer_guid, const **SequenceNumber** &the_sequence_number)
  *Create a **SampleIdentity** (p. 2282) with the provided **Guid** (p. 1596) and **SequenceNumber** (p. 2335).*
- const **Guid** & **writer_guid** () const
  *Get the 16-byte identifier identifying the virtual **Guid** (p. 1596).*
- **Guid** & **writer_guid** ()
  *Get the 16-byte identifier identifying the virtual **Guid** (p. 1596).*
- const **SequenceNumber** & **sequence_number** () const
  *Get the monotonically increasing 64-bit **SequenceNumber** (p. 2335) that identifies the sample in the data source.*
- **SequenceNumber** & **sequence_number** ()
  *Get the monotonically increasing 64-bit **SequenceNumber** (p. 2335) that identifies the sample in the data source.*

Static Public Member Functions

- static **SampleIdentity** **automatic** ()
  Special value to indicate that RTI Connext will automatically assign the identity of a sample.
- static **SampleIdentity** **unknown** ()
  An invalid or unknown sample identity.

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator**<< (std::ostream &out, const **SampleIdentity** &sid)
  *Prints a **SampleIdentity** (p. 2282) to an output stream.*

8.329.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* A **SampleIdentity** (p. 2282) defines a pair (Virtual Writer **Guid** (p. 1596), **SequenceNumber** (p. 2335)) that uniquely identifies a sample within a DDS domain and a Topic.

8.329.2 Constructor & Destructor Documentation

8.329.2.1 SampleIdentity() [1/2]

```
rti::core::SampleIdentity::SampleIdentity ( )  [inline]
```

Create a default **SampleIdentity** (p. 2282) object.

8.329.2.2 SampleIdentity() [2/2]

```
rti::core::SampleIdentity::SampleIdentity (
            const  Guid & the_writer_guid,
            const  SequenceNumber & the_sequence_number )  [inline]
```

Create a **SampleIdentity** (p. 2282) with the provided **Guid** (p. 1596) and **SequenceNumber** (p. 2335).

Parameters

the_writer_guid	The **Guid** (p. 1596) to create the **SampleIdentity** (p. 2282) with
the_sequence_number	The **SequenceNumber** (p. 2335) to create the **SampleIdentity** (p. 2282) with

8.329.3 Member Function Documentation

8.329.3.1 writer_guid() [1/2]

```
const  Guid & rti::core::SampleIdentity::writer_guid ( ) const  [inline]
```

Get the 16-byte identifier identifying the virtual **Guid** (p. 1596).

8.329.3.2 writer_guid() [2/2]

```
Guid & rti::core::SampleIdentity::writer_guid ( )  [inline]
```

Get the 16-byte identifier identifying the virtual **Guid** (p. 1596).

8.329.3.3 sequence_number() [1/2]

```
const  SequenceNumber & rti::core::SampleIdentity::sequence_number ( ) const  [inline]
```

Get the monotonically increasing 64-bit **SequenceNumber** (p. 2335) that identifies the sample in the data source.

8.329.3.4 sequence_number() [2/2]

```
SequenceNumber & rti::core::SampleIdentity::sequence_number ( )  [inline]
```

Get the monotonically increasing 64-bit **SequenceNumber** (p. 2335) that identifies the sample in the data source.

8.329.3.5 automatic()

```
static  SampleIdentity rti::core::SampleIdentity::automatic ( )  [inline], [static]
```

Special value to indicate that RTI Connext will automatically assign the identity of a sample.

8.329.3.6 unknown()

```
static  SampleIdentity rti::core::SampleIdentity::unknown ( )  [inline], [static]
```

An invalid or unknown sample identity.

8.329.4 Friends And Related Function Documentation

8.329.4.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const  SampleIdentity & sid )  [related]
```

Prints a **SampleIdentity** (p. 2282) to an output stream.

8.330 dds::sub::SampleInfo Class Reference

<<***value-type***>> *(p. 350)* Information that accompanies each sample received by a **DataReader** (p. 976)

```
#include "dds/sub/SampleInfo.hpp"
```

Public Member Functions

- **dds::core::Time source_timestamp** () const

  Get the timestamp when the sample was written by a DataWriter.
- **dds::sub::status::DataState state** () const

  *Get the **dds::sub::status::DataState** (p. 1118) of the sample.*
- **dds::sub::GenerationCount generation_count** () const

  *Get the **dds::sub::GenerationCount** (p. 1589) of the sample.*
- **dds::sub::Rank rank** () const

  *Get the **dds::sub::Rank** (p. 2130) of the sample.*
- bool **valid** () const

  *Indicates whether the DataSample contains data or else it is only used to communicate a change in the **dds::sub::status::InstanceState** (p. 1615) of the instance.*
- **dds::core::InstanceHandle instance_handle** () const

  Identifies locally the corresponding instance.
- **dds::core::InstanceHandle publication_handle** () const

  Identifies locally the DataWriter that modified the instance.
- **dds::core::Time reception_timestamp** () const

  <<***extension***>> *(p. 354) The timestamp when the sample was committed by a **dds::sub::DataReader** (p. 976).*
- **rti::core::SequenceNumber publication_sequence_number** () const

  <<***extension***>> *(p. 354) The publication sequence number.*
- **rti::core::SequenceNumber reception_sequence_number** () const

  <<***extension***>> *(p. 354) The reception sequence number when sample was committed by a **dds::sub::DataReader** (p. 976)*
- **rti::core::Guid original_publication_virtual_guid** () const

  <<***extension***>> *(p. 354) The original publication virtual GUID.*
- **rti::core::SequenceNumber original_publication_virtual_sequence_number** () const

  <<***extension***>> *(p. 354) The original publication virtual sequence number.*
- **rti::core::SampleIdentity original_publication_virtual_sample_identity** () const

  <<***extension***>> *(p. 354) Retrieves the information provided by **original_publication_virtual_guid()** (p. 2291) and **original_publication_virtual_sequence_number()** (p. 2291) combined in a SampleIdentity instance*
- **rti::core::Guid related_original_publication_virtual_guid** () const

<<***extension***>> *(p. 354) The original publication virtual GUID of a related sample*

- **rti::core::SequenceNumber related_original_publication_virtual_sequence_number** () const

  <<***extension***>> *(p. 354) The original publication virtual sequence number of a related sample*
- **rti::core::SampleIdentity related_original_publication_virtual_sample_identity** () const

  <<***extension***>> *(p. 354) Retrieves the information provided by **related_original_publication_virtual_guid()** (p. 2292) and **related_original_publication_virtual_sequence_number()** (p. 2292) combined in a SampleIdentity instance*
- **rti::core::SampleFlag flag** () const

  <<***extension***>> *(p. 354) Flags associated with the sample.*
- **rti::core::Guid source_guid** () const

  <<***extension***>> *(p. 354) The application logical data source associated with the sample.*
- **rti::core::Guid related_source_guid** () const

  <<***extension***>> *(p. 354) The application logical data source that is related to the sample.*
- **rti::core::Guid related_subscription_guid** () const

  <<***extension***>> *(p. 354) The related_reader_guid associated with the sample.*
- **rti::core::Guid topic_query_guid** () const

  <<***extension***>> *(p. 354) The GUID of the **rti::sub::TopicQuery** (p. 2516) that is related to the sample.*
- rti::core::EncapsulationId **encapsulation_id** () const

  <<***extension***>> *(p. 354) The encapsulation kind.*
- **rti::core::optional_value**< **rti::core::CoherentSetInfo** > **coherent_set_info** () const

  <<***extension***>> *(p. 354) When set, this field provides the information about the coherent set associated with the sample.*

8.330.1 Detailed Description

<<***value-type***>> *(p. 350)* Information that accompanies each sample received by a **DataReader** (p. 976)

Information that accompanies each sample that is `read` or `taken`.

8.330.2 Interpretation of the SampleInfo

The **dds::sub::SampleInfo** (p. 2285) contains information pertaining to the associated `Data` instance sample including:

- the **dds::sub::status::DataState** (p. 1118) of the Data value. The DataState provides information about:
- the `sample_state` of the `Data` value (i.e., if it has already been read or not)
- the `view_state` of the related instance (i.e., if the instance is new or not)
- the `instance_state` of the related instance (i.e., if the instance is alive or not)
- **dds::sub::SampleInfo::valid** (p. 2289) flag. This flag indicates whether there is data associated with the sample. Some samples do not contain data indicating only a change on the `instance_state` of the corresponding instance.

- the **dds::sub::GenerationCount** (p. 1589) of the Data value. The **GenerationCount** (p. 1589) provides information about:

- The values of `disposed_generation_count` and `no_writers_generation_count` for the related instance at the time the sample was received. These counters indicate the number of times the instance had become ALIVE (with `instance_state=` **dds::sub::status::InstanceState::alive()** (p. 1617)) at the time the sample was received.

- the **dds::sub::Rank** (p. 2130) of the Data value. The **Rank** (p. 2130) provides information about:

- The `sample_rank` and `generation_rank` of the sample within the returned sequence. These ranks provide a preview of the samples that follow within the sequence returned by the `read` or `take` operations.

- The `absolute_generation_rank` of the sample within the **dds::sub::DataReader** (p. 976). This rank provides a preview of what is available within the **dds::sub::DataReader** (p. 976).

- The `source_timestamp` of the sample. This is the timestamp provided by the **dds::pub::DataWriter** (p. 1138) at the time the sample was produced.

8.330.3 Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count

For each instance, RTI Connext internally maintains two counts, the **dds::sub::GenerationCount::disposed()** (p. 1590) and **dds::sub::GenerationCount::no_writers()** (p. 1590), relative to each **DataReader** (p. 976):

- The **dds::sub::GenerationCount::disposed()** (p. 1590) and **dds::sub::GenerationCount::no_writers()** (p. 1590) are initialized to zero when the **dds::sub::DataReader** (p. 976) first detects the presence of a never-seen-before instance.

- The **dds::sub::GenerationCount::disposed()** (p. 1590) is incremented each time the instance_state of the corresponding instance changes from **dds::sub::status::InstanceState::not_alive_disposed()** (p. 1617) to **dds::sub::status::InstanceState::alive()** (p. 1617).

- The **dds::sub::GenerationCount::no_writers()** (p. 1590) is incremented each time the instance_state of the corresponding instance changes from **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) to **dds::sub::status::InstanceState::alive()** (p. 1617).

- These 'generation counts' are reset to zero when the instance resource is reclaimed.

The **dds::sub::GenerationCount::disposed()** (p. 1590) and **dds::sub::GenerationCount::no_writers()** (p. 1590) available in the **dds::sub::SampleInfo** (p. 2285) capture a snapshot of the corresponding counters at the time the sample was received.

8.330.4 Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank

The **dds::sub::Rank::sample()** (p. 2132) and **dds::sub::Rank::generation()** (p. 2132) available in the **dds::sub::SampleInfo** (p. 2285) are computed based solely on the actual samples in the ordered collection returned by read or take.

- The **dds::sub::Rank::sample()** (p. 2132) indicates the number of samples of the same instance that follow the current one in the collection.
- The **dds::sub::Rank::generation()** (p. 2132) available in the **dds::sub::SampleInfo** (p. 2285) indicates the difference in "generations" between the sample (S) and the Most Recent **Sample** (p. 2270) of the same instance that appears in the returned Collection (MRSIC). That is, it counts the number of times the instance transitioned from not-alive to alive in the time from the reception of the S to the reception of MRSIC.
- These 'generation ranks' are reset to zero when the instance resource is reclaimed.

The **dds::sub::Rank::generation()** (p. 2132) is computed using the formula:

```
generation_rank = (MRSIC.disposed_generation_count
                       + MRSIC.no_writers_generation_count)
                  - (S.disposed_generation_count
                       + S.no_writers_generation_count)
```

The **dds::sub::Rank::absolute_generation()** (p. 2132) available in the **dds::sub::SampleInfo** (p. 2285) indicates the difference in "generations" between the sample (S) and the Most Recent **Sample** (p. 2270) of the same instance that the middleware has received (MRS). That is, it counts the number of times the instance transitioned from not-alive to alive in the time from the reception of the S to the time when the read or take was called.

```
absolute_generation_rank = (MRS.disposed_generation_count
                                + MRS.no_writers_generation_count)
                           - (S.disposed_generation_count
                                + S.no_writers_generation_count)
```

8.330.5 Interpretation of the SampleInfo counters and ranks

These counters and ranks allow the application to distinguish samples belonging to different "generations" of the instance. Note that it is possible for an instance to transition from not-alive to alive (and back) several times before the application accesses the data by means of read or take. In this case, the returned collection may contain samples that cross generations (i.e. some samples were received before the instance became not-alive, other after the instance re-appeared again). Using the information in the **dds::sub::SampleInfo** (p. 2285), the application can anticipate what other information regarding the same instance appears in the returned collection, as well as in the infrastructure and thus make appropriate decisions.

For example, an application desiring to only consider the most current sample for each instance would only look at samples with sample_rank == 0. Similarly, an application desiring to only consider samples that correspond to the latest generation in the collection will only look at samples with generation_rank == 0. An application desiring only samples pertaining to the latest generation available will ignore samples for which absolute_generation_rank != 0. Other application-defined criteria may also be used.

See also

dds::sub::status::SampleState (p. 2316), **dds::sub::status::InstanceState** (p. 1615), **dds::sub::status::ViewState** (p. 2614), **dds::sub::SampleInfo::valid** (p. 2289)

"Statechart of the \p instance_state and \p view_state of a single instance"

8.330.6 Member Function Documentation

8.330.6.1 source_timestamp()

```
dds::core::Time dds::sub::SampleInfo::source_timestamp ( ) const  [inline]
```

Get the timestamp when the sample was written by a DataWriter.

8.330.6.2 state()

```
dds::sub::status::DataState dds::sub::SampleInfo::state ( ) const  [inline]
```

Get the **dds::sub::status::DataState** (p. 1118) of the sample.

The DataState encapsulates the sample's **dds::sub::status::InstanceState** (p. 1615), **dds::sub::status::ViewState** (p. 2614), and **dds::sub::status::SampleState** (p. 2316)

8.330.6.3 generation_count()

```
dds::sub::GenerationCount dds::sub::SampleInfo::generation_count ( ) const  [inline]
```

Get the **dds::sub::GenerationCount** (p. 1589) of the sample.

8.330.6.4 rank()

```
dds::sub::Rank dds::sub::SampleInfo::rank ( ) const  [inline]
```

Get the **dds::sub::Rank** (p. 2130) of the sample.

8.330.6.5 valid()

```
bool dds::sub::SampleInfo::valid ( ) const  [inline]
```

Indicates whether the DataSample contains data or else it is only used to communicate a change in the **dds::sub::status::InstanceState** (p. 1615) of the instance.

Normally each `DataSample` contains both a **dds::sub::SampleInfo** (p. 2285) and some Data. However there are situations where a `DataSample` contains only the **dds::sub::SampleInfo** (p. 2285) and does not have any associated data. This occurs when the RTI Connext notifies the application of a change of state for an instance that was caused by some internal mechanism (such as a timeout) for which there is no associated data. An example of this situation is when the RTI Connext detects that an instance has no writers and changes the corresponding instance_state to **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617).

The application can distinguish whether a particular `DataSample` has data by examining the value of the **`dds::sub::SampleInfo::valid`** (p. 2289). If this flag is set to true, then the `DataSample` contains valid Data. If the flag is set to false, the `DataSample` contains no Data.

Applications should check the **valid()** (p. 2289) flag before accessing the data. An attempt to access **rti::sub::LoanedSample::data()** (p. 1661) or **dds::sub::Sample::data()** (p. 2272) when **valid()** (p. 2289) is false will throw **dds::core::PreconditionNotMetError** (p. 1941).

See also

Foo_subscriber.cxx (p. **??**)

rti::sub::LoanedSample::info() (p. 1661)

dds::sub::Sample::info() (p. 2273)

Referenced by **rti::sub::LoanedSample< T >::operator<<()**.

8.330.6.6 instance_handle()

```
dds::core::InstanceHandle dds::sub::SampleInfo::instance_handle ( ) const  [inline]
```

Identifies locally the corresponding instance.

The handle is equal to **dds::core::InstanceHandle::nil()** (p. 1614) for unkeyed topics.

8.330.6.7 publication_handle()

```
dds::core::InstanceHandle dds::sub::SampleInfo::publication_handle ( ) const  [inline]
```

Identifies locally the DataWriter that modified the instance.

The `publication_handle` is the same **dds::core::InstanceHandle** (p. 1612) that is returned by the operation **dds::sub::matched_publications** (p. 668) and can also be used as a parameter to the operation **dds::sub::DataReader::matched_publication_data** (p. 1033).

8.330.6.8 reception_timestamp()

```
dds::core::Time reception_timestamp ( ) const
```

<<***extension***>> *(p. 354)* The timestamp when the sample was committed by a **dds::sub::DataReader** (p. 976).

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.330.6.9 publication_sequence_number()

```
rti::core::SequenceNumber publication_sequence_number ( ) const
```

<<***extension***>> *(p. 354)* The publication sequence number.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.330.6.10 reception_sequence_number()

```
rti::core::SequenceNumber reception_sequence_number ( ) const
```

<<***extension***>> *(p. 354)* The reception sequence number when sample was committed by a **dds::sub::DataReader** (p. 976)

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.330.6.11 original_publication_virtual_guid()

```
rti::core::Guid original_publication_virtual_guid ( ) const
```

<<***extension***>> *(p. 354)* The original publication virtual GUID.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

If the **dds::core::policy::Presentation::access_scope** (p. 1947) of the **dds::pub::Publisher** (p. 1993) is **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951), this field contains the **dds::pub::Publisher** (p. 1993) virtual GUID that uniquely identifies the DataWriter group.

See also

Sample::identity

8.330.6.12 original_publication_virtual_sequence_number()

```
rti::core::SequenceNumber original_publication_virtual_sequence_number ( ) const
```

<<***extension***>> *(p. 354)* The original publication virtual sequence number.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

If the **dds::core::policy::Presentation::access_scope** (p. 1947) of the **dds::pub::Publisher** (p. 1993) is **dds::core::policy::PresentationAccessScopeKind_def::GROUP** (p. 1951), this field contains the **dds::pub::Publisher** (p. 1993) virtual sequence number that uniquely identifies a sample within the DataWriter group.

See also

Sample::identity

8.330.6.13 original_publication_virtual_sample_identity()

```
rti::core::SampleIdentity original_publication_virtual_sample_identity ( ) const
```

<<***extension***>> *(p. 354)* Retrieves the information provided by **original_publication_virtual_guid()** (p. 2291) and **original_publication_virtual_sequence_number()** (p. 2291) combined in a SampleIdentity instance

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Referenced by **rti::request::Replier< RequestType, ReplyType >::send_reply()**.

8.330.6.14 related_original_publication_virtual_guid()

```
rti::core::Guid related_original_publication_virtual_guid ( ) const
```

<<***extension***>> *(p. 354)* The original publication virtual GUID of a related sample

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.330.6.15 related_original_publication_virtual_sequence_number()

```
rti::core::SequenceNumber related_original_publication_virtual_sequence_number ( ) const
```

<<***extension***>> *(p. 354)* The original publication virtual sequence number of a related sample

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.330.6.16 related_original_publication_virtual_sample_identity()

```
rti::core::SampleIdentity related_original_publication_virtual_sample_identity ( ) const
```

<<***extension***>> *(p. 354)* Retrieves the information provided by **related_original_publication_virtual_guid()** (p. 2292) and **related_original_publication_virtual_sequence_number()** (p. 2292) combined in a SampleIdentity instance

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Referenced by **rti::request::IsReplyRelatedPredicate< T >::operator()()**.

8.330.6.17 flag()

```
rti::core::SampleFlag flag ( ) const
```

<<***extension***>> *(p. 354)* Flags associated with the sample.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The flags can be set by using the field **rti::pub::WriteParams::flag** (p. 2649) when writing a sample using the method **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179).

8.330.6.18 source_guid()

```
rti::core::Guid source_guid ( ) const
```

<<***extension***>> *(p. 354)* The application logical data source associated with the sample.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The source_guid can be set by using the field **rti::pub::WriteParams::source_guid** (p. 2650) when writing a sample using the method **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179).

See also

rti::pub::WriteParams::source_guid (p. 2650) for additional information on **source_guid** (p. 2293) and when it is used.

8.330.6.19 related_source_guid()

```
rti::core::Guid related_source_guid ( ) const
```

<<***extension***>> *(p. 354)* The application logical data source that is related to the sample.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The source_guid can be set by using the field **rti::pub::WriteParams::source_guid** (p. 2650) when writing a sample using the method **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179).

See also

rti::pub::WriteParams::source_guid (p. 2650) for additional information on **source_guid** (p. 2293) and when it is used.

8.330.6.20 related_subscription_guid()

```
rti::core::Guid related_subscription_guid ( ) const
```

<<***extension***>> *(p. 354)* The related_reader_guid associated with the sample.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

The related_reader_guid can be set by using the field **rti::pub::WriteParams::related_reader_guid** (p. 2651) when writing a sample using the method **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179).

8.330.6.21 topic_query_guid()

```
rti::core::Guid topic_query_guid ( ) const
```

<<***extension***>> *(*p. *354)* The GUID of the **rti::sub::TopicQuery** (p. 2516) that is related to the sample.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

This GUID indicates whether a sample is part of the response to a **rti::sub::TopicQuery** (p. 2516) or a regular ("live") sample:

- If the sample was written for the TopicQuery stream, this field contains the GUID of the target TopicQuery.
- If the sample was written for the live stream, this field will be set to **rti::core::Guid::unknown()** (p. 1597).

8.330.6.22 encapsulation_id()

```
rti::core::EncapsulationId encapsulation_id ( ) const
```

<<***extension***>> *(*p. *354)* The encapsulation kind.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.330.6.23 coherent_set_info()

```
rti::core::optional_value< rti::core::CoherentSetInfo > coherent_set_info ( ) const
```

<<***extension***>> *(*p. *354)* When set, this field provides the information about the coherent set associated with the sample.

This field is set for all samples that are part of a coherent set. Coherent sets are initiated using the operation **dds::pub::CoherentSet::CoherentSet** (p. 932) and finalized using the operation **dds::pub::CoherentSet::end()** (p. 933).

See also

dds::pub::CoherentSet::CoherentSet (p. 932) for additional information on coherent sets.

8.331 rti::sub::SampleIterator< T > Class Template Reference

A random-access iterator of **LoanedSample** (p. 1659).

`#include <SampleIterator.hpp>`

Public Member Functions

- **SampleIterator** & **operator++** ()

8.331.1 Detailed Description

template<typename T>
class rti::sub::SampleIterator< T >

A random-access iterator of **LoanedSample** (p. 1659).

See also

dds::sub::LoanedSamples (p. 1663)

8.331.2 Member Function Documentation

8.331.2.1 operator++()

```
template<typename T >
SampleIterator &  rti::sub::SampleIterator< T >::operator++ ( )  [inline]
```

Coverity complains about this increment because it may overflow. However the user is responsible for not incrementing the iterator beyond the end of the sequence when accessing the iterator. With the current implementation we are providing a big enough iterator to iterate over a full sequence without overflowing.

8.332 rti::core::status::SampleLostState Class Reference

<<***extension***>> *(*p. *354)* Reasons why a sample was lost

`#include <Status.hpp>`

Inherits std::bitset< OMG_DDS_STATE_BIT_COUNT >.

Public Member Functions

- **SampleLostState** ()

  *Creates **not_lost()** (p. 2299)*
- **SampleLostState** (const MaskType &src)

  Create from a bitset.

Static Public Member Functions

- static const **SampleLostState not_lost** ()

  The sample was not lost.
- static const **SampleLostState lost_by_writer** ()

  *A **dds::pub::DataWriter** (p. 1138) removed the sample before being received by the **dds::sub::DataReader** (p. 976).*
- static const **SampleLostState lost_by_instances_limit** ()

  *A resource limit on the number of instances (**dds::core::policy::ResourceLimits::max_instances** (p. 2206)) was reached.*
- static const **SampleLostState lost_by_remote_writers_per_instance_limit** ()

  *A resource limit on the number of remote writers for a single instance from which a **dds::sub::DataReader** (p. 976) may read (**rti::core::policy::DataReaderResourceLimits::max_remote_writers_per_instance** (p. 1086)) was reached.*
- static const **SampleLostState lost_by_incomplete_coherent_set** ()

  A sample is lost because it is part of an incomplete coherent set. An incomplete coherent set is a coherent set for which some of the samples are missing.
- static const **SampleLostState lost_by_large_coherent_set** ()

  *A sample is lost because it is part of a large coherent set. A large coherent set is a coherent set that cannot fit all at once into the **dds::sub::DataReader** (p. 976) queue because resource limits are exceeded.*
- static const **SampleLostState lost_by_samples_per_remote_writer_limit** ()

  *When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157): a resource limit on the number of samples from a given remote writer that a **dds::sub::DataReader** (p. 976) may store (**rti::core::policy::DataReader**↩ **ResourceLimits::max_samples_per_remote_writer** (p. 1087)) was reached. When using **dds::core::policy::**↩ **ReliabilityKind_def::RELIABLE** (p. 2157), reaching **rti::core::policy::DataReaderResourceLimits::max_samples**↩ **_per_remote_writer** (p. 1087) will trigger a rejection, not a loss, with reason **dds::core::status::SampleRejected**↩ **State::rejected_by_samples_per_remote_writer_limit()** (p. 2313).*
- static const **SampleLostState lost_by_virtual_writers_limit** ()

  *A resource limit on the number of virtual writers from which a **dds::sub::DataReader** (p. 976) may read (**rti::core**↩ **::policy::DataReaderResourceLimits::max_remote_virtual_writers** (p. 1097)) was reached.*
- static const **SampleLostState lost_by_remote_writers_per_sample_limit** ()

  *A resource limit on the number of remote writers per sample (**rti::core::policy::DataReaderResourceLimits::max_**↩ **remote_writers_per_sample** (p. 1099)) was reached.*
- static const **SampleLostState lost_by_availability_waiting_time** ()

  ***rti::core::policy::Availability::max_data_availability_waiting_time** (p. 871) expired.*
- static const **SampleLostState lost_by_remote_writers_per_virtual_queue_limit** ()

  *A resource limit on the number of samples published by a remote writer on behalf of a virtual writer that a **dds::sub::**↩ **DataReader** (p. 976) may store was reached. (This field is currently not used.)*
- static const **SampleLostState lost_by_out_of_memory** ()

  A sample was lost because there was not enough memory to store the sample.
- static const **SampleLostState lost_by_unknown_instance** ()

  A received sample was lost because it doesn't contain enough information for the reader to know what instance it is associated with.
- static const **SampleLostState lost_by_deserialization_failure** ()

A received sample was lost because it could not be deserialized.

- static const **SampleLostState lost_by_decode_failure** ()

  *When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157): A received sample was lost because it could not be decoded. When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157), the sample will be rejected, not lost, with reason **dds::core::status::SampleRejectedState::rejected_by_decode_failure()** (p. 2314).*

- static const **SampleLostState lost_by_samples_per_instance_limit** ()

  *When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157): A resource limit on the number of samples per instance (**dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207)) was reached. When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157), reaching **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207) will trigger a rejection, not a loss, with reason **dds::core::status::SampleRejectedState::rejected_by_samples_per_instance_limit()** (p. 2313).*

- static const **SampleLostState lost_by_samples_limit** ()

  *When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157): A resource limit on the number of samples (**dds::core::policy::ResourceLimits::max_samples** (p. 2205)) was reached. When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157), reaching **dds::core::policy::ResourceLimits::max_samples** (p. 2205) will trigger a rejection, not a loss, with reason **dds::core::status::SampleRejectedState::rejected_by_samples_limit()** (p. 2312).*

8.332.1 Detailed Description

<<***extension***>> *(p. 354)* Reasons why a sample was lost

See also

dds::core::status::SampleLostStatus::last_reason() (p. 2304)

8.332.2 Constructor & Destructor Documentation

8.332.2.1 SampleLostState() [1/2]

```
rti::core::status::SampleLostState::SampleLostState ( )  [inline]
```

Creates **not_lost()** (p. 2299)

8.332.2.2 SampleLostState() [2/2]

```
rti::core::status::SampleLostState::SampleLostState (
            const MaskType & src )  [inline]
```

Create from a bitset.

8.332.3 Member Function Documentation

8.332.3.1 not_lost()

```
static const SampleLostState rti::core::status::SampleLostState::not_lost ( ) [inline], [static]
```

The sample was not lost.

8.332.3.2 lost_by_writer()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_writer ( ) [inline],
[static]
```

A **dds::pub::DataWriter** (p. 1138) removed the sample before being received by the **dds::sub::DataReader** (p. 976).

8.332.3.3 lost_by_instances_limit()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_instances_limit ( )
[inline], [static]
```

A resource limit on the number of instances (**dds::core::policy::ResourceLimits::max_instances** (p. 2206)) was reached.

See also

dds::core::policy::ResourceLimits (p. 2203)

8.332.3.4 lost_by_remote_writers_per_instance_limit()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_remote_writers_per_↩
instance_limit ( ) [inline], [static]
```

A resource limit on the number of remote writers for a single instance from which a **dds::sub::DataReader** (p. 976) may read (**rti::core::policy::DataReaderResourceLimits::max_remote_writers_per_instance** (p. 1086)) was reached.

See also

rti::core::policy::DataReaderResourceLimits (p. 1081)

8.332.3.5 lost_by_incomplete_coherent_set()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_incomplete_coherent_set
( ) [inline], [static]
```

A sample is lost because it is part of an incomplete coherent set. An incomplete coherent set is a coherent set for which some of the samples are missing.

For example, consider a **dds::pub::DataWriter** (p. 1138) using **dds::core::policy::HistoryKind_def::KEEP_LAST** (p. 1607) with a depth of 1. The DataWriter publishes two samples of the same instance as part of a coherent set 'CS1'; the first sample of 'CS1' is replaced by a new sample before it can be successfully delivered to the **dds::sub::DataReader** (p. 976). In this case, the coherent set containing the two samples is considered incomplete. The new sample, by default, will not be provided to the application, and will be reported as LOST_BY_INCOMPLETE_COHERENT_SET. (You can change this default behavior by setting **dds::core::policy::Presentation::drop_incomplete_coherent_set** (p. 1949) to FALSE. If you do, the new sample will be provided to the application, but it will be marked as part of an incomplete coherent set in the **dds::sub::SampleInfo** (p. 2285) structure.)

8.332.3.6 lost_by_large_coherent_set()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_large_coherent_set ( )
[inline], [static]
```

A sample is lost because it is part of a large coherent set. A large coherent set is a coherent set that cannot fit all at once into the **dds::sub::DataReader** (p. 976) queue because resource limits are exceeded.

For example, if **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207) on the DataReader is 10 and the coherent set has 15 samples for a given instance, the coherent set is a large coherent set that will be considered incomplete.

The resource limits that can lead to large coherent sets are: **dds::core::policy::ResourceLimits::max_samples** (p. 2205), **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207), **dds::core::policy::ResourceLimits::max_instances** (p. 2206), and **rti::core::policy::DataReaderResourceLimits::max_samples_per_remote_writer** (p. 1087).

8.332.3.7 lost_by_samples_per_remote_writer_limit()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_samples_per_remote_
writer_limit ( ) [inline], [static]
```

When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157): a resource limit on the number of samples from a given remote writer that a **dds::sub::DataReader** (p. 976) may store (**rti::core::policy::DataReaderResourceLimits::max_samples_per_remote_writer** (p. 1087)) was reached. When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157), reaching **rti::core::policy::DataReaderResourceLimits::max_samples_per_remote_writer** (p. 1087) will trigger a rejection, not a loss, with reason **dds::core::status::SampleRejectedState::rejected_by_samples_per_remote_writer_limit()** (p. 2313).

See also

rti::core::policy::DataReaderResourceLimits (p. 1081)

8.332.3.8 lost_by_virtual_writers_limit()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_virtual_writers_limit (
) [inline], [static]
```

A resource limit on the number of virtual writers from which a **dds::sub::DataReader** (p. 976) may read (**rti::core↩ ::policy::DataReaderResourceLimits::max_remote_virtual_writers** (p. 1097)) was reached.

See also

rti::core::policy::DataReaderResourceLimits (p. 1081)

8.332.3.9 lost_by_remote_writers_per_sample_limit()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_remote_writers_per_↩
sample_limit ( ) [inline], [static]
```

A resource limit on the number of remote writers per sample (**rti::core::policy::DataReaderResourceLimits::max_↩ remote_writers_per_sample** (p. 1099)) was reached.

See also

rti::core::policy::DataReaderResourceLimits (p. 1081)

8.332.3.10 lost_by_availability_waiting_time()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_availability_waiting_↩
time ( ) [inline], [static]
```

rti::core::policy::Availability::max_data_availability_waiting_time (p. 871) expired.

See also

rti::core::policy::Availability (p. 868)

8.332.3.11 lost_by_remote_writers_per_virtual_queue_limit()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_remote_writers_per_↩
virtual_queue_limit ( ) [inline], [static]
```

A resource limit on the number of samples published by a remote writer on behalf of a virtual writer that a **dds::sub::↩ DataReader** (p. 976) may store was reached. (This field is currently not used.)

See also

rti::core::policy::DataReaderResourceLimits (p. 1081)

8.332.3.12 lost_by_out_of_memory()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_out_of_memory ( ) [inline],
[static]
```

A sample was lost because there was not enough memory to store the sample.

See also

rti::core::policy::DataReaderResourceLimits (p. 1081)

8.332.3.13 lost_by_unknown_instance()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_unknown_instance ( )
[inline], [static]
```

A received sample was lost because it doesn't contain enough information for the reader to know what instance it is associated with.

8.332.3.14 lost_by_deserialization_failure()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_deserialization_failure
( ) [inline], [static]
```

A received sample was lost because it could not be deserialized.

8.332.3.15 lost_by_decode_failure()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_decode_failure ( ) [inline],
[static]
```

When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157): A received sample was lost because it could not be decoded. When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157), the sample will be rejected, not lost, with reason **dds::core::status::SampleRejectedState::rejected_by_decode_failure()** (p. 2314).

8.332.3.16 lost_by_samples_per_instance_limit()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_samples_per_instance_↩
limit ( ) [inline], [static]
```

When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157): A resource limit on the number of samples per instance (**dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207)) was reached. When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157), reaching **dds::core::policy::Resource↩ Limits::max_samples_per_instance** (p. 2207) will trigger a rejection, not a loss, with reason **dds::core::status::↩ SampleRejectedState::rejected_by_samples_per_instance_limit()** (p. 2313).

See also

dds::core::policy::ResourceLimits (p. 2203)

8.332.3.17 lost_by_samples_limit()

```
static const SampleLostState rti::core::status::SampleLostState::lost_by_samples_limit ( ) [inline],
[static]
```

When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157): A resource limit on the number of samples (**dds::core::policy::ResourceLimits::max_samples** (p. 2205)) was reached. When using **dds::core↩ ::policy::ReliabilityKind_def::RELIABLE** (p. 2157), reaching **dds::core::policy::ResourceLimits::max_samples** (p. 2205) will trigger a rejection, not a loss, with reason **dds::core::status::SampleRejectedState::rejected_by_↩ samples_limit()** (p. 2312).

See also

dds::core::policy::ResourceLimits (p. 2203)

8.333 dds::core::status::SampleLostStatus Class Reference

Information about the status **dds::core::status::StatusMask::sample_lost()** (p. 2381)

```
#include <TStatus.hpp>
```

Public Member Functions

- int32_t **total_count** () const

  *Total cumulative count of all samples lost across all instances of data published under the **dds::topic::Topic** (p. 2474).*
- int32_t **total_count_change** () const

  The incremental number of samples lost since the last time the listener was called or the status was read.
- **rti::core::status::SampleLostState last_reason** () const

  *<<**extension**>> (p. 354) Reason why the last sample was lost.*

8.333.1 Detailed Description

Information about the status **dds::core::status::StatusMask::sample_lost()** (p. 2381)

8.333.2 Member Function Documentation

8.333.2.1 total_count()

```
int32_t dds::core::status::SampleLostStatus::total_count ( ) const  [inline]
```

Total cumulative count of all samples lost across all instances of data published under the **dds::topic::Topic** (p. 2474).

8.333.2.2 total_count_change()

```
int32_t dds::core::status::SampleLostStatus::total_count_change ( ) const  [inline]
```

The incremental number of samples lost since the last time the listener was called or the status was read.

8.333.2.3 last_reason()

```
rti::core::status::SampleLostState last_reason ( ) const
```

*<<**extension**>> (p. 354)* Reason why the last sample was lost.

See also

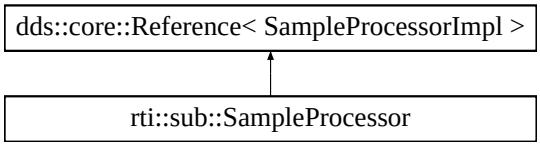
rti::core::status::SampleLostState (p. 2296)

8.334 rti::sub::SampleProcessor Class Reference

*<<**extension**>> (p. 354) <<**reference-type**>> (p. 351)* Utility to read and process the data samples that one or more DataReaders receive using a sample handler.

```
#include <rti/sub/SampleProcessor.hpp>
```

Inheritance diagram for rti::sub::SampleProcessor:



Public Member Functions

- **SampleProcessor** (const **rti::core::cond::AsyncWaitSetProperty** &aws_property)

  *Single-argument constructor that allows creating a a **rti::sub::SampleProcessor** (p. 2305) with the configuration of the underlying **rti::core::cond::AsyncWaitSet** (p. 840).*
- **SampleProcessor** ()

  *Creates a **rti::sub::SampleProcessor** (p. 2305) with the default **rti::core::cond::AsyncWaitSetProperty** (p. 858).*
- **SampleProcessor** (**rti::core::cond::AsyncWaitSet** aws)

  *Constructor that allows specifying an externally created **rti::core::cond::AsyncWaitSet** (p. 840).*
- template<typename T , typename FUNCTOR >
  SampleProcessor & **attach_reader** (**dds::sub::DataReader**< T > reader, const FUNCTOR &handler)

  *Attaches the specified **dds::sub::DataReader** (p. 976) with an associated sample handler to this **rti::sub::SampleProcessor** (p. 2305).*
- template<typename T >
  SampleProcessor & **detach_reader** (**dds::sub::DataReader**< T > reader)

  *Detaches the specified **dds::sub::DataReader** (p. 976) from this **rti::sub::SampleProcessor** (p. 2305).*
- std::vector< **dds::sub::AnyDataReader** > **readers** () const

  *Returns the list of attached **dds::sub::DataReader** (p. 976) (s).*

8.334.1 Detailed Description

*<<**extension**>> (p. 354) <<**reference-type**>> (p. 351)* Utility to read and process the data samples that one or more DataReaders receive using a sample handler.

The following example shows how to create a full application that subscribes to a topic and prints each sample it receives:

```
int subscriber_main(int domain_id, int sample_count)
{
     dds::domain::DomainParticipant participant(domain_id);
     dds::topic::Topic<Foo> topic(participant, "Example Foo");
     dds::sub::DataReader<Foo> reader(dds::sub::Subscriber(participant), topic);
     int count = 0;
     rti::sub::SampleProcessor sample_processor;
     sample_processor.attach_reader(reader, [](const rti::sub::LoanedSample<Foo>& sample) {
         if (sample.info().valid()) {
             std::cout « "Received " « sample.data() « std::endl;
```

```
        }
    });
    while(1) {
        // wait forever
    }
    return 1;
}
```

Note that the expected sample handler is a functor with signature std::function<void(const rti::sub::LoanedSample<T>&)>, where `T` is the type associated with the **dds::sub::DataReader** (p. 976).

A **rti::sub::SampleProcessor** (p. 2305) relies on an underlying **rti::core::cond::AsyncWaitSet** (p. 840) to read and dispatch the data from the DataReaders. It internally creates a **dds::sub::cond::ReadCondition** (p. 2133) for each attached DataReader, an associates a custom handler that contains state to read samples and notify the corresponding handler .

The **SampleProcessor** (p. 2305) uses this ReadCondition to wait for data and then calls **dds::sub::DataReader::select()** (p. 1003) to take all the available data.

Notifications to any handler may be concurrent if the thread pool size set in **rti::core::cond::AsyncWaitSetProperty::thread_pool_size** (p. 862) is greater than 1. The same or different handlers may be called in parallel. The **SampleProcessor** (p. 2305) cycles through all attached DataReaders dispatching a sample at a time with the next available thread. This mechanism guarantees concurrent dispatching of the samples accross all DataReaders.

The **rti::sub::SampleProcessor** (p. 2305) internally creates and uses a **dds::sub::cond::ReadCondition** (p. 2133) for each attached **dds::sub::DataReader** (p. 976) to read the data. It's recommended to perform all the reading of the **dds::sub::DataReader** (p. 976)'s samples through the **rti::sub::SampleProcessor** (p. 2305) and avoid reading through other mechanisms in different parts of your applications. Similarly, your application must be careful to not modify the **dds::sub::cond::ReadCondition** (p. 2133) associated with an attached **dds::sub::DataReader** (p. 976).

In general, avoid or beware of using the following operations in combination with a **rti::sub::SampleProcessor** (p. 2305):

- read or take operation (and any of its overloads) of a **dds::sub::DataReader** (p. 976) from any part of your application, including a **dds::sub::DataReaderListener** (p. 1055) or a handler .
- Perform actions based on the **dds::core::cond::StatusCondition** (p. 2372) with the **dds::core::status::StatusMask::data_available()** (p. 2383) enabled (for example through a **rti::core::cond::AsyncWaitSet** (p. 840)).

On the other hand, you can externally provide the underlying **rti::sub::SampleProcessor** (p. 2305)'s **rti::core::cond::AsyncWaitSet** (p. 840) if you want to access additional capabilities of the **rti::core::cond::AsyncWaitSet** (p. 840) and use it to attach other **dds::core::cond::Condition** (p. 948) to handle other aspects of your application.

8.334.2 SampleProcessor Thread Safety

Similar to the **rti::core::cond::AsyncWaitSet** (p. 840), the **rti::sub::SampleProcessor** (p. 2305) provides a thread-safe interface. All the operations of this class can be called concurrently from multiple threads.

Because the **rti::sub::SampleProcessor** (p. 2305) relies on an **rti::core::cond::AsyncWaitSet** (p. 840) to dispatch the samples the same threading concepts apply. This means that operations on a **rti::sub::SampleProcessor** (p. 2305) may require synchronizing with the thread pool for safety. **rti::sub::SampleProcessor** (p. 2305) also relies on the asynchrnous completion pattern to effectively interact with the underlying thread pool.

For instance to detach a **dds::sub::DataReader** (p. 976), the **rti::sub::SampleProcessor** (p. 2305) generates an internal request to its thread pool to process it. As soon as the detachment completes, the thread pool provides the notification through an associated completion token on which the **rti::sub::SampleProcessor** (p. 2305) waits and blocks until it completes.

Due to the concurrent processing nature of operations on a **rti::sub::SampleProcessor** (p. 2305) as well as the sample handling, it's important to keep in mind the following aspects:

- The handler operation can be called concurrently for each sample. Therefore, handler implementations may need to apply thread synchronization strategies to protect shared resources.
- Calling a **rti::sub::SampleProcessor** (p. 2305) operation that requires internal synchronization (e.g., **rti::sub↩ ::SampleProcessor::attach_reader** (p. 2309)) from a handler notification will result in a error.

Note that the interface of the **rti::sub::SampleProcessor** (p. 2305) is similar to that of the **rti::core::cond::Async↩ WaitSet** (p. 840) but reduced and simplified. As noted above, if more advanced use and control of the thread pool is required, you can always create the **rti::core::cond::AsyncWaitSet** (p. 840) externally when calling dds::sub::Sample↩ Processor::SampleProcessor(rti::core::cond::AsyncWaitSet).

MT Safety:

Safe.

See also

rti::core::cond::AsyncWaitSet (p. 840)

dds::core::cond::Condition (p. 948)

rti::core::cond::AsyncWaitSetProperty (p. 858)

dds::sub::DataReader (p. 976)

8.334.3 Constructor & Destructor Documentation

8.334.3.1 SampleProcessor() [1/3]

```
rti::sub::SampleProcessor::SampleProcessor (
            const  rti::core::cond::AsyncWaitSetProperty & aws_property )  [inline]
```

Single-argument constructor that allows creating a a **rti::sub::SampleProcessor** (p. 2305) with the configuration of the underlying **rti::core::cond::AsyncWaitSet** (p. 840).

You can provide **rti::core::cond::AsyncWaitSetProperty::AsyncWaitSetProperty()** (p. 859) as `property` to create the underlying **rti::core::cond::AsyncWaitSet** (p. 840) with default behavior.

This constructor creates **rti::core::cond::AsyncWaitSet** (p. 840) with no listener installed and will call **rti::core::cond↩ ::AsyncWaitSet::start** (p. 845) rigt after its creation.

Parameters

aws_property	<<***in***>> *(*p. *355)* configuration of the underlying **rti::core::cond::AsyncWaitSet** (p. 840).

See also

dds::sub::SampleProcessor::SampleProcessor(rti::core::cond::AsyncWaitSet)

8.334.3.2 SampleProcessor() [2/3]

```
rti::sub::SampleProcessor::SampleProcessor ( )  [inline]
```

Creates a **rti::sub::SampleProcessor** (p. 2305) with the default **rti::core::cond::AsyncWaitSetProperty** (p. 858).

See also

rti::core::cond::AsyncWaitSetProperty::AsyncWaitSetProperty() (p. 859)

8.334.3.3 SampleProcessor() [3/3]

```
rti::sub::SampleProcessor::SampleProcessor (
              rti::core::cond::AsyncWaitSet aws )  [inline]
```

Constructor that allows specifying an externally created **rti::core::cond::AsyncWaitSet** (p. 840).

Creates a new **rti::sub::SampleProcessor** (p. 2305) with the specified **rti::core::cond::AsyncWaitSet** (p. 840).

Tnis constructor flavor decouples the lifecycle of the **rti::sub::SampleProcessor** (p. 2305) from the **rti::core::cond::↩ AsyncWaitSet** (p. 840). It allows the application to have more control on the **rti::core::cond::AsyncWaitSet** (p. 840), such as to install an **rti::core::cond::AsyncWaitSetListener** (p. 856), a custom ThreadFactory, and controlling when it starts or stops.

Note that this constructor will not call **rti::core::cond::AsyncWaitSet::start** (p. 845), so it's the caller responsiblity to start and stop it. You can provide the external **rti::core::cond::AsyncWaitSet::start** (p. 845) in either started or stopped state.

Parameters

aws	<<***in***>> *(*p. *355)* the externally created **rti::core::cond::AsyncWaitSet** (p. 840).

See also

dds::sub::SampleProcessor::SampleProcessor(const rti::core::cond::AsyncWaitSetProperty&)

8.334.4 Member Function Documentation

8.334.4.1 attach_reader()

```
template<typename T , typename FUNCTOR >
SampleProcessor & rti::sub::SampleProcessor::attach_reader (
             dds::sub::DataReader< T > reader,
            const FUNCTOR & handler )  [inline]
```

Attaches the specified **dds::sub::DataReader** (p. 976) with an associated sample handler to this **rti::sub::SampleProcessor** (p. 2305).

Note that the expected sample handler is a functor with signature std::function<void(const rti::sub::LoanedSample<T>&)>, where `T` is the type associated with the **dds::sub::DataReader** (p. 976).

This operation will block until the attach request completes. Upon successful return, it is guaranteed that the specified **dds::sub::DataReader** (p. 976) is attached and notifications to the handler may occur.

If this operation is called multiple times for an already attached **dds::sub::DataReader** (p. 976), it will result in no-op and return sucessfully, ignoring the specified handler. So if the handler is different, it will not be updated.

Parameters

reader	<<***in***>> *(p. 355)* **dds::sub::DataReader** (p. 976) to be attached.
handler	<<***in***>> *(p. 355)* handler to be notified on dispatching of the `reader` samples.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::sub::SampleProcessor::detach_reader (p. 2309)

References **attach_reader()**.

Referenced by **attach_reader()**.

8.334.4.2 detach_reader()

```
template<typename T >
SampleProcessor & rti::sub::SampleProcessor::detach_reader (
            dds::sub::DataReader< T > reader ) [inline]
```

Detaches the specified **dds::sub::DataReader** (p. 976) from this **rti::sub::SampleProcessor** (p. 2305).

Once the **dds::sub::DataReader** (p. 976) is detached, it is guaranteed that the **rti::sub::SampleProcessor** (p. 2305) will no longer process it so it is safe for your application to release any resources associated with the detached **dds::sub::DataReader** (p. 976).

This operation blocks until the detach request completes. Upon successful return, it is guaranteed that the specified **dds::sub::DataReader** (p. 976) is detached.

dds::sub::DataReader (p. 976) may be detached at any time independently of the state of the **rti::sub::SampleProcessor** (p. 2305).

Parameters

reader	<<***in***>> *(p. 355)* **dds::sub::DataReader** (p. 976) to be detached.

Exceptions

One	of the **Standard Exceptions** (p. 428)

See also

rti::sub::SampleProcessor::attach_reader (p. 2309)

References **detach_reader()**.

Referenced by **detach_reader()**.

8.334.4.3 readers()

```
std::vector< dds::sub::AnyDataReader > rti::sub::SampleProcessor::readers ( ) const [inline]
```

Returns the list of attached **dds::sub::DataReader** (p. 976) (s).

8.335 dds::core::status::SampleRejectedState Class Reference

Reasons why a sample was rejected.

```
#include <State.hpp>
```

Inherits std::bitset< OMG_DDS_STATE_BIT_COUNT >.

Public Member Functions

- **SampleRejectedState** ()

  Creates not_lost()
- **SampleRejectedState** (const MaskType &src)

  Create from a bitset.

Static Public Member Functions

- static const **SampleRejectedState not_rejected** ()

  The sample was not rejected.
- static const **SampleRejectedState rejected_by_samples_limit** ()

  *When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157): A resource limit on the number of samples (**dds::core::policy::ResourceLimits::max_samples** (p. 2205)) was reached. When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157), reaching **dds::core::policy::ResourceLimits::max_samples** (p. 2205) will trigger a loss, not a rejection, with reason **rti::core::status::SampleLostState::lost_by_samples_limit()** (p. 2303).*
- static const **SampleRejectedState rejected_by_instances_limit** ()

  *Connext DDS does not reject samples based on instance limits (**dds::core::policy::ResourceLimits::max_instances** (p. 2206)), so this value will never be used.*
- static const **SampleRejectedState rejected_by_samples_per_instance_limit** ()

  *When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157): A resource limit on the number of samples per instance (**dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207)) was reached. When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157), reaching **dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207) will trigger a loss, not a rejection, with reason **rti::core::status::SampleLostState::lost_by_samples_per_instance_limit()** (p. 2303).*
- static const **SampleRejectedState rejected_by_samples_per_remote_writer_limit** ()

  *When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157): a resource limit on the number of samples from a given remote writer that a **dds::sub::DataReader** (p. 976) may store (**rti::core::policy::DataReaderResourceLimits::max_samples_per_remote_writer** (p. 1087)) was reached. When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157), reaching **rti::core::policy::DataReaderResourceLimits::max_samples_per_remote_writer** (p. 1087) will trigger a loss, not a rejection, with reason **rti::core::status::SampleLostState::lost_by_samples_per_remote_writer_limit()** (p. 2300).*
- static const **SampleRejectedState rejected_by_remote_writers_per_virtual_queue_limit** ()

  *A resource limit on the number of samples published by a remote writer on behalf of a virtual writer that a **dds::sub::DataReader** (p. 976) may store was reached. (This field is currently not used.)*
- static const **SampleRejectedState rejected_by_decode_failure** ()

  *When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157): A received sample was rejected because it could not be decoded. When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157), the sample will be lost, not rejected, with reason **rti::core::status::SampleLostState::lost_by_decode_failure()** (p. 2302).*

8.335.1 Detailed Description

Reasons why a sample was rejected.

See also

dds::core::status::SampleRejectedStatus::last_reason() (p. 2315)

8.335.2 Constructor & Destructor Documentation

8.335.2.1 SampleRejectedState() [1/2]

```
dds::core::status::SampleRejectedState::SampleRejectedState ( )  [inline]
```

Creates not_lost()

Referenced by **not_rejected()**, **rejected_by_decode_failure()**, **rejected_by_instances_limit()**, **rejected_by_remote_writers_per_virtual_queue_limit()**, **rejected_by_samples_limit()**, **rejected_by_samples_per_instance_limit()**, and **rejected_by_samples_per_remote_writer_limit()**.

8.335.2.2 SampleRejectedState() [2/2]

```
dds::core::status::SampleRejectedState::SampleRejectedState (
            const MaskType & src )  [inline]
```

Create from a bitset.

8.335.3 Member Function Documentation

8.335.3.1 not_rejected()

```
static const  SampleRejectedState dds::core::status::SampleRejectedState::not_rejected ( )  [inline],
[static]
```

The sample was not rejected.

References **SampleRejectedState()**.

8.335.3.2 rejected_by_samples_limit()

```
static const SampleRejectedState dds::core::status::SampleRejectedState::rejected_by_samples_↩
limit ( ) [inline], [static]
```

When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157): A resource limit on the number of samples (**dds::core::policy::ResourceLimits::max_samples** (p. 2205)) was reached. When using **dds::core::policy↩ ::ReliabilityKind_def::BEST_EFFORT** (p. 2157), reaching **dds::core::policy::ResourceLimits::max_samples** (p. 2205) will trigger a loss, not a rejection, with reason **rti::core::status::SampleLostState::lost_by_samples↩ _limit()** (p. 2303).

See also

dds::core::policy::ResourceLimits (p. 2203)

References **SampleRejectedState()**.

8.335.3.3 rejected_by_instances_limit()

```
static const SampleRejectedState dds::core::status::SampleRejectedState::rejected_by_instances_↩
limit ( ) [inline], [static]
```

Connext DDS does not reject samples based on instance limits (**dds::core::policy::ResourceLimits::max_instances** (p. 2206)), so this value will never be used.

See also

rti::core::status::SampleLostState::lost_by_instances_limit() (p. 2299)

References **SampleRejectedState()**.

8.335.3.4 rejected_by_samples_per_instance_limit()

```
static const SampleRejectedState dds::core::status::SampleRejectedState::rejected_by_samples_↩
per_instance_limit ( ) [inline], [static]
```

When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157): A resource limit on the number of samples per instance (**dds::core::policy::ResourceLimits::max_samples_per_instance** (p. 2207)) was reached. When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157), reaching **dds::core::policy::ResourceLimits↩ ::max_samples_per_instance** (p. 2207) will trigger a loss, not a rejection, with reason **rti::core::status::Sample↩ LostState::lost_by_samples_per_instance_limit()** (p. 2303).

See also

ResourceLimitsQosPolicy

References **SampleRejectedState()**.

8.335.3.5 rejected_by_samples_per_remote_writer_limit()

```
static const SampleRejectedState dds::core::status::SampleRejectedState::rejected_by_samples_↩
per_remote_writer_limit ( ) [inline], [static]
```

When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157): a resource limit on the number of samples from a given remote writer that a **dds::sub::DataReader** (p. 976) may store (**rti::core::policy::DataReaderResource↩ Limits::max_samples_per_remote_writer** (p. 1087)) was reached. When using **dds::core::policy::Reliability↩ Kind_def::BEST_EFFORT** (p. 2157), reaching **rti::core::policy::DataReaderResourceLimits::max_samples_per↩ _remote_writer** (p. 1087) will trigger a loss, not a rejection, with reason **rti::core::status::SampleLostState::lost_↩ by_samples_per_remote_writer_limit()** (p. 2300).

See also

rti::core::policy::DataReaderResourceLimits (p. 1081)

References **SampleRejectedState()**.

8.335.3.6 rejected_by_remote_writers_per_virtual_queue_limit()

```
static const SampleRejectedState dds::core::status::SampleRejectedState::rejected_by_remote_↩
writers_per_virtual_queue_limit ( ) [inline], [static]
```

A resource limit on the number of samples published by a remote writer on behalf of a virtual writer that a **dds::sub::↩ DataReader** (p. 976) may store was reached. (This field is currently not used.)

See also

rti::core::policy::DataReaderResourceLimits (p. 1081)

References **SampleRejectedState()**.

8.335.3.7 rejected_by_decode_failure()

```
static const SampleRejectedState dds::core::status::SampleRejectedState::rejected_by_decode_↩
failure ( ) [inline], [static]
```

When using **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157): A received sample was rejected because it could not be decoded. When using **dds::core::policy::ReliabilityKind_def::BEST_EFFORT** (p. 2157), the sample will be lost, not rejected, with reason **rti::core::status::SampleLostState::lost_by_decode_failure()** (p. 2302).

If a sample was rejected for this reason and the **dds::pub::DataWriter** (p. 1138) set **rti::core::policy::Data↩ WriterProtocol::disable_inline_keyhash** (p. 1214) to true, then **dds::core::status::SampleRejectedStatus::last↩ _instance_handle** (p. 2316) may not be correct if the sample was encrypted.

References **SampleRejectedState()**.

8.336 dds::core::status::SampleRejectedStatus Class Reference

Information about the status **dds::core::status::StatusMask::sample_rejected()** (p. 2382)

`#include <TStatus.hpp>`

Public Member Functions

- int32_t **total_count** () const

  *Total cumulative count of samples rejected by the **dds::sub::DataReader** (p. 976).*
- int32_t **total_count_change** () const

  The incremental number of samples rejected since the last time the listener was called or the status was read.
- const **dds::core::status::SampleRejectedState last_reason** () const

  Reason for rejecting the last sample rejected.
- const **dds::core::InstanceHandle last_instance_handle** () const

  Handle to the instance being updated by the last sample that was rejected.

8.336.1 Detailed Description

Information about the status **dds::core::status::StatusMask::sample_rejected()** (p. 2382)

8.336.2 Member Function Documentation

8.336.2.1 total_count()

```
int32_t dds::core::status::SampleRejectedStatus::total_count ( ) const  [inline]
```

Total cumulative count of samples rejected by the **dds::sub::DataReader** (p. 976).

8.336.2.2 total_count_change()

```
int32_t dds::core::status::SampleRejectedStatus::total_count_change ( ) const  [inline]
```

The incremental number of samples rejected since the last time the listener was called or the status was read.

8.336.2.3 last_reason()

```
const dds::core::status::SampleRejectedState dds::core::status::SampleRejectedStatus::last_reason
( ) const [inline]
```

Reason for rejecting the last sample rejected.

See also

dds::core::status::SampleRejectedState (p. 2310)

8.336.2.4 last_instance_handle()

```
const dds::core::InstanceHandle dds::core::status::SampleRejectedStatus::last_instance_handle ( )
const [inline]
```

Handle to the instance being updated by the last sample that was rejected.

If the sample was rejected because of **dds::core::status::SampleRejectedState::rejected_by_decode_failure()** (p. 2314) and the **dds::pub::DataWriter** (p. 1138) set **rti::core::policy::DataWriterProtocol::disable_inline_keyhash** (p. 1214) to true, then the last_instance_handle may not be correct if the sample was encrypted.

8.337 dds::sub::status::SampleState Class Reference

Indicates whether or not a sample has ever been read.

```
#include <dds/sub/status/DataState.hpp>
```

Inherits std::bitset< OMG_DDS_STATE_BIT_COUNT >.

Public Types

- typedef std::bitset< OMG_DDS_STATE_BIT_COUNT > **MaskType**

  An std::bitset of SampleStates.

Public Member Functions

- **SampleState** (const **MaskType** &other)

  *Create an **SampleState** (p. 2316) from MaskType.*

Static Public Member Functions

- static const **SampleState** **read** ()

  *Creates a read **SampleState** (p. 2316).*
- static const **SampleState** **not_read** ()

  *Creates a not_read **SampleState** (p. 2316) object.*
- static const **SampleState** **any** ()

  *Creates a **SampleState** (p. 2316) object representing any sample state.*

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator**<< (std::ostream &os, const **SampleState** &s)

  Prints a sample state as a readable string.

8.337.1 Detailed Description

Indicates whether or not a sample has ever been read.

For example, you can check if a sample has been read before as follows:

```
for (auto sample : reader.read()) {
    if (sample.info().state().sample_state() ==
            dds::sub::status::SampleState::read()) {
        // ...
    }
}
```

8.337.2 Member Typedef Documentation

8.337.2.1 MaskType

```
typedef std::bitset<OMG_DDS_STATE_BIT_COUNT >  dds::sub::status::SampleState::MaskType
```

An std::bitset of SampleStates.

8.337.3 Constructor & Destructor Documentation

8.337.3.1 SampleState()

```
dds::sub::status::SampleState::SampleState (
            const  MaskType & other )  [inline]
```

Create an **SampleState** (p. 2316) from MaskType.

Parameters

other	The MaskType to create the **SampleState** (p. 2316) with

8.337.4 Member Function Documentation

8.337.4.1 read()

```
static const SampleState dds::sub::status::SampleState::read ( ) [inline], [static]
```

Creates a read **SampleState** (p. 2316).

Indicates that the application has already accessed that sample by means of a read or take operation.

8.337.4.2 not_read()

```
static const SampleState dds::sub::status::SampleState::not_read ( ) [inline], [static]
```

Creates a not_read **SampleState** (p. 2316) object.

Indicates that the application has not accessed that sample before.

Referenced by **dds::sub::status::DataState::new_data()**.

8.337.4.3 any()

```
static const SampleState dds::sub::status::SampleState::any ( ) [inline], [static]
```

Creates a **SampleState** (p. 2316) object representing any sample state.

This is equivalent to **read()** (p. 2318) | **not_read()** (p. 2318)

Referenced by **dds::sub::status::DataState::any()**, **dds::sub::status::DataState::any_data()**, and **dds::sub::status::DataState::new_instance()**.

8.337.5 Friends And Related Function Documentation

8.337.5.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const  SampleState & s )  [related]
```

Prints a sample state as a readable string.

8.338 rti::config::ScopedLoggerVerbosity Class Reference

Changes the logger verbosity temporarily during the scope of a variable.

```
#include <rti/config/Logger.hpp>
```

Public Member Functions

- **ScopedLoggerVerbosity** (**Verbosity** verbosity)
- ~**ScopedLoggerVerbosity** ()

8.338.1 Detailed Description

Changes the logger verbosity temporarily during the scope of a variable.

RAII utility to modify logging verbosity during the scope of a variable and restore it to the previous value upon destruction.

8.338.2 Constructor & Destructor Documentation

8.338.2.1 ScopedLoggerVerbosity()

```
rti::config::ScopedLoggerVerbosity::ScopedLoggerVerbosity (
             Verbosity verbosity )  [inline]
```

Sets the verbosity of **Logger::instance()** (p. 1685)

References **rti::config::Logger::instance()**, and **rti::config::Logger::verbosity()**.

8.338.2.2 ~ScopedLoggerVerbosity()

```
rti::config::ScopedLoggerVerbosity::~ScopedLoggerVerbosity ( )  [inline]
```

Restores the verbosity of **Logger::instance()** (p. 1685) to the value before the construction of this object.

References **rti::config::Logger::instance()**, and **rti::config::Logger::verbosity()**.

8.339 dds::sub::DataReader< T >::Selector Class Reference

The **Selector** (p. 2320) class is used by the **DataReader** (p. 976) to compose read and take operations.

```
#include <TDataReader.hpp>
```

Public Member Functions

- **Selector** (**DataReader** &dr)

  *Create a **Selector** (p. 2320) for this **DataReader** (p. 976).*
- **Selector** & **instance** (const **dds::core::InstanceHandle** &h)

  Select a specific instance to read/take.
- **Selector** & **next_instance** (const **dds::core::InstanceHandle** &previous_handle)

  Select the instance after a specific instance.
- **Selector** & **state** (const **dds::sub::status::DataState** &s)

  Select a specific DataState.
- **Selector** & **content** (const **dds::sub::Query** &query)

  *Select samples based on a **Query** (p. 2053).*
- **Selector** & **condition** (const **dds::sub::cond::ReadCondition** &the_condition)

  Select samples based on a ReadCondition.
- **Selector** & **max_samples** (int32_t n)

  Choose to only read/take up to a maximum number of samples.
- **dds::sub::LoanedSamples**< T > **read** ()

  *Read samples based on the state associated with this **Selector** (p. 2320).*
- **dds::sub::LoanedSamples**< T > **take** ()

  *Take samples based on the state associated with this **Selector** (p. 2320).*
- **DataReader** **reader** () const

  *Returns the **DataReader** (p. 976) associated with the **Selector** (p. 2320).*
- template<typename SamplesFWIterator >
  uint32_t **read** (SamplesFWIterator sfit, int32_t the_max_samples)

  Read up to max_samples samples, inserting the samples into a provided container.
- template<typename SamplesFWIterator >
  uint32_t **take** (SamplesFWIterator sfit, int32_t the_max_samples)

  Take up to max_samples samples, inserting the samples into a provided container.
- template<typename SamplesBIIterator >
  uint32_t **read** (SamplesBIIterator sbit)

  Read samples, inserting them into a provided container.
- template<typename SamplesBIIterator >
  uint32_t **take** (SamplesBIIterator sbit)

  Take samples, inserting them into a provided container.

8.339.1 Detailed Description

template<typename T>
class dds::sub::DataReader< T >::Selector

The **Selector** (p. 2320) class is used by the **DataReader** (p. 976) to compose read and take operations.

A **Selector** (p. 2320) has an associated **DataReader** (p. 976) and configures the behavior of the read or take operation performed by that **DataReader** (p. 976).

A **Selector** (p. 2320) call is typically composed of the following steps:

1. Start with a call to `reader.select()` to return a **Selector** (p. 2320);
2. Follow with one or more **Selector** (p. 2320) setters configure what data to read;
3. Finally call **`read()`** (p. 2325) or **`take()`** (p. 2326) to perform the operation.

The following example reads a maximum of 5 previously unread samples:

```
LoanedSamples<Foo> samples = reader.select()
      .max_samples(5)
      .state(dds::sub::status::DataState::new_data());
      .read()
```

The selector also allows iterating over the samples one instance at a time, using the **next_instance()** (p. 2322) setter.

See also

Accessing Received Data (p. 318)

8.339.2 Constructor & Destructor Documentation

8.339.2.1 Selector()

```
template<typename T >
dds::sub::DataReader< T >::Selector::Selector (
              DataReader & dr )  [inline]
```

Create a **Selector** (p. 2320) for this **DataReader** (p. 976).

Selectors are created with the default filter state of the **DataReader** (p. 976)

Parameters

dr	The **DataReader** (p. 976) that this **Selector** (p. 2320) is attached to

See also

dds::sub::DataReader::default_filter_state() (p. 991)

8.339.3 Member Function Documentation

8.339.3.1 instance()

```
template<typename T >
Selector &  dds::sub::DataReader< T >::Selector::instance (
             const  dds::core::InstanceHandle & h )  [inline]
```

Select a specific instance to read/take.

This operation causes the subsequent read or take operation to access only samples belonging the single specified instance whose handle is h.

Upon successful completion, the data collection will contain samples all belonging to the same instance. The corresponding **SampleInfo** (p. 2285) verifies **SampleInfo.instance_handle()** (p. 2290) == h.

The subsequent read/take operation will be semantically equivalent to a read or take without specifying the instance, except in building the collection, the **DataReader** (p. 976) will check that the sample belongs to the specified instance and otherwise it will not place the sample in the returned collection.

The subsequent read/take may operation may fail with **dds::core::InvalidArgumentError** (p. 1619) if the Instance↩ Handle does not correspond to an existing data-object known to the **DataReader** (p. 976).

Parameters

h	The handle of the instance to select

8.339.3.2 next_instance()

```
template<typename T >
Selector &  dds::sub::DataReader< T >::Selector::next_instance (
             const  dds::core::InstanceHandle & previous_handle )  [inline]
```

Select the instance after a specific instance.

This operation causes the subsequent read or take operation to access only samples belonging a single instance whose handle is considered 'next' after the provided InstanceHandle h.

The accessed samples will all belong to the 'next' instance with InstanceHandle 'greater' than the specified previous handle that has available samples.

This operation implies the existence of a total order 'greater-than' relationship between the instance handles. The specifics of this relationship are not all important and are implementation specific. The important thing is that, according to the middleware, all instances are ordered relative to each other. This ordering is between the instance handles; It should not depend on the state of the instance (e.g. whether it has data or not) and must be defined even for instance handles that do not correspond to instances currently managed by the **dds::sub::DataReader** (p. 976). For the purposes of the ordering, it should be 'as if' each instance handle was represented as unique integer.

The behavior of **dds::sub::DataReader::Selector::next_instance** (p. 2322) is 'as if' the **dds::sub::DataReader** (p. 976) invoked **dds::sub::instance(const dds::core::InstanceHandle& h)** (p. 665), passing the smallest instance_handle among all the ones that: (a) are greater than `previous_handle`, and (b) have available samples (i.e. samples that meet the constraints imposed by the specified states).

The special value **dds::core::InstanceHandle::nil()** (p. 1614) is guaranteed to be 'less than' any valid `instance_handle`. So the use of the parameter value `previous_handle` == **dds::core::InstanceHandle::nil()** (p. 1614) will return the samples for the instance which has the smallest `instance_handle` among all the instances that contain available samples.

Note

The operation **dds::sub::DataReader::Selector::next_instance** (p. 2322) is intended to be used in an application-driven iteration, where the application starts by passing `previous_handle` == **dds::core::InstanceHandle::nil()** (p. 1614), examines the samples returned, and then uses the `instance_handle` returned in the **dds::sub::SampleInfo** (p. 2285) as the value of the `previous_handle` argument to the next call to **dds::sub::DataReader::Selector::next_instance** (p. 2322). The iteration continues until the read/take operation doesn't return any more samples. This application-driven iteration is required to ensure that all samples on the reader queue are read.

Note that it is possible to call the **dds::sub::DataReader::Selector::next_instance** (p. 2322) operation with a `previous_handle` that does not correspond to an instance currently managed by the **dds::sub::DataReader** (p. 976). This is because as stated earlier the 'greater-than' relationship is defined even for handles not managed by the **dds::sub::DataReader** (p. 976). One practical situation where this may occur is when an application is iterating though all the instances, takes all the samples of a **dds::sub::status::InstanceState::not_alive_no_writers()** (p. 1617) instance, returns the loan (at which point the instance information may be removed, and thus the handle becomes invalid), and tries to read the next instance.

Parameters

previous_handle	The 'next smallest' instance with a value greater than this value that has available samples will be returned.

See also

Selecting what samples to read (p. 319) for an example

8.339.3.3 state()

```
template<typename T >
Selector &  dds::sub::DataReader< T >::Selector::state (
              const  dds::sub::status::DataState & s )  [inline]
```

Select a specific DataState.

By setting the DataState, you can specify the state of the samples that should be read or taken. The DataState of a sample encapsulates the SampleState, ViewState, and InstanceState of a sample. For example:

```
using namespace dds::sub::status;
LoanedSamples<Foo> samples = reader.select()
      .state(DataState(SampleState::not_read(), ViewState::new_view(), InstanceState::alive())
      .read();
```

If you're only interested in the SampleState, the ViewState, or the InstanceState, you can pass it directly. For example, to read alive instances with any SampleState or ViewState:

```
LoanedSamples<Foo> samples = reader.select()
      .state(InstanceState::alive())
      .read();
```

If this method comes before a call to **condition()** (p. 2324), then it will be overridden and will not have any effect on the read or take operation.

Parameters

s	The selected DataState

8.339.3.4 content()

```
template<typename T >
Selector &  dds::sub::DataReader< T >::Selector::content (
             const  dds::sub::Query & query )  [inline]
```

Select samples based on a **Query** (p. 2053).

The effect of using this manipulator is that the subsequent read/take will filter the samples based on the **Query** (p. 2053)'s expression. If the **DataReader** (p. 976) has no samples that meet the constraints, the read/take will not return any data.

If this method is called before a call to condition then it will be overridden and will not have any effect on the read or take operation. Similarly, if this operation follows a call to **condition()** (p. 2324), then the previously set ReadCondition will be cleared.

Parameters

query	The **Query** (p. 2053) to read/take with

8.339.3.5 condition()

```
template<typename T >
Selector &  dds::sub::DataReader< T >::Selector::condition (
             const  dds::sub::cond::ReadCondition & the_condition )  [inline]
```

Select samples based on a ReadCondition.

When passing a QueryCondition, the effect of calling this method is that the subsequent read/take will filter the samples based on the QueryConditions's expression and state. If the **DataReader** (p. 976) has no samples that meet the constraints, the read/take will not return any data.

Passing an actual ReadCondition will only filter based on the state.

If this method is called before a call to content then it will be overridden and will not have any effect on the read or take operation. Similarly, if this operation follows a call to **content()** (p. 2324) and/or state, then the previously set **Query** (p. 2053) and DataState will be cleared.

This method is effectively a combination of calling content and state, but a QueryCondition is more efficient when reading with the same query multiple times.

For example:
```
samples = reader.select()
                .read()
                .content(dds::sub::Query(system.reader, "foo = 7"))
                .state(dds::sub::status::DataState::new_data())
```

is equivalent to:
```
samples = reader.select()
                .read()
                .condition(QueryCondition(Query(system.reader, "foo = 7"), DataState::new_data()))
```

Parameters

the_condition	The ReadCondition (or QueryCondition) to read/take with

8.339.3.6 max_samples()

```
template<typename T >
Selector &  dds::sub::DataReader< T >::Selector::max_samples (
              int32_t n )  [inline]
```

Choose to only read/take up to a maximum number of samples.

Parameters

n	The maximum number of samples to read/take

8.339.3.7 read() [1/3]

```
template<typename T >
dds::sub::LoanedSamples< T >  dds::sub::DataReader< T >::Selector::read ( )  [inline]
```

Read samples based on the state associated with this **Selector** (p. 2320).

Returns

dds::sub::LoanedSamples<T>

8.339.3.8 take() [1/3]

```
template<typename T >
dds::sub::LoanedSamples< T >  dds::sub::DataReader< T >::Selector::take ( )  [inline]
```

Take samples based on the state associated with this **Selector** (p. 2320).

Returns

dds::sub::LoanedSamples<T>

8.339.3.9 reader()

```
template<typename T >
DataReader  dds::sub::DataReader< T >::Selector::reader ( ) const  [inline]
```

Returns the **DataReader** (p. 976) associated with the **Selector** (p. 2320).

Returns

dds::sub::DataReader (p. 976)

8.339.3.10 read() [2/3]

```
template<typename T >
template<typename SamplesFWIterator >
uint32_t  dds::sub::DataReader< T >::Selector::read (
             SamplesFWIterator sfit,
             int32_t the_max_samples )  [inline]
```

Read up to max_samples samples, inserting the samples into a provided container.

Template Parameters

SamplesFWIterator	A forward iterator whose value type is dds::sub::Sample<T>

Parameters

sfit	A forward iterator pointing to the position in a container in which to put the read samples
the_max_samples	The maximum number of samples to read

Returns

uint32_t The number of read samples

8.339.3.11 take() [2/3]

```
template<typename T >
template<typename SamplesFWIterator >
uint32_t  dds::sub::DataReader< T >::Selector::take (
             SamplesFWIterator sfit,
             int32_t the_max_samples )  [inline]
```

Take up to max_samples samples, inserting the samples into a provided container.

Template Parameters

SamplesFWIterator	A forward iterator whose value type is dds::sub::Sample<T>

Parameters

sfit	A forward iterator pointing to the position in a container in which to put the taken samples
the_max_samples	The maximum number of samples to take

Returns

uint32_t The number of taken samples

8.339.3.12 read() [3/3]

```
template<typename T >
template<typename SamplesBIIterator >
uint32_t  dds::sub::DataReader< T >::Selector::read (
             SamplesBIIterator sbit )  [inline]
```

Read samples, inserting them into a provided container.

Template Parameters

SamplesBIIterator	A back-inserting iterator whose value type is dds::sub::Sample<T>

Parameters

sbit	A back-inserting iterator placed at the position in the destination container where the read samples should be placed

Returns

uint32_t The number of samples that were read

8.339.3.13 take() [3/3]

```
template<typename T >
template<typename SamplesBIIterator >
uint32_t  dds::sub::DataReader< T >::Selector::take (
             SamplesBIIterator sbit )  [inline]
```

Take samples, inserting them into a provided container.

Template Parameters

SamplesBIIterator	A back-inserting iterator whose value type is dds::sub::Sample<T>

Parameters

sbit	A back-inserting iterator placed at the position in the destination container where the taken samples should be placed

Returns

uint32_t The number of samples that were taken

8.340 rti::flat::SequenceIterator< E, OffsetKind > Class Template Reference

Iterator for collections of Offsets.

```
#include <SequenceIterator.hpp>
```

Public Types

- typedef std::forward_iterator_tag **iterator_category**

  The iterator category.
- typedef E **value_type**

  The element type.
- typedef **value_type** **reference**

  The reference type is the same as the value type, an Offset.
- typedef **value_type** **pointer**

  The pointer type is the same as the value type, an Offset.
- typedef std::ptrdiff_t **difference_type**

  The difference type.

Public Member Functions

- **SequenceIterator** ()

  Constructs an invalid iterator.
- bool **is_null** () const

  Returns whether the iterator is invalid.
- **value_type** **operator*** () const

  Returns the Offset of the current element.
- **value_type** **operator->** () const

  Returns the Offset of the current element.
- bool **advance** ()

  Advances to the next element, reporting any errors by returning false.
- **SequenceIterator** & **operator++** ()

  Advances to the next element.
- **SequenceIterator** **operator++** (int)

  Advances to the next element.

Friends

- bool **operator<** (const **SequenceIterator** &s1, const **SequenceIterator** &s2)

  Compares two iterators.
- bool **operator>** (const **SequenceIterator** &s1, const **SequenceIterator** &s2)

  Compares two iterators.
- bool **operator<=** (const **SequenceIterator** &s1, const **SequenceIterator** &s2)

  Compares two iterators.
- bool **operator>=** (const **SequenceIterator** &s1, const **SequenceIterator** &s2)

  Compares two iterators.
- bool **operator==** (const **SequenceIterator** &s1, const **SequenceIterator** &s2)

  Compares two iterators.
- bool **operator!=** (const **SequenceIterator** &s1, const **SequenceIterator** &s2)

  Compares two iterators.

8.340.1 Detailed Description

template<typename E, typename OffsetKind>
class rti::flat::SequenceIterator< E, OffsetKind >

Iterator for collections of Offsets.

Template Parameters

E	The Offset type of the elements
OffsetKind	(implementation detail)

Note

This type should not be declared directly. For example, for a **SequenceOffset** (p. 2343), use **SequenceOffset::iterator** (p. 782) (or simply `auto`).

A **SequenceIterator** (p. 2328) moves through the Offset to elements of a sequence or array **Offset** (p. 414), and provides the functionality of a standard `forward_iterator`.

Note that a dereferenced **SequenceIterator** (p. 2328) returns **by value** the Offset to the current element; it doesn't return it by reference.

8.340.2 Member Typedef Documentation

8.340.2.1 iterator_category

```
template<typename E , typename OffsetKind >
typedef std::forward_iterator_tag  rti::flat::SequenceIterator< E, OffsetKind >::iterator_category
```

The iterator category.

8.340.2.2 value_type

```
template<typename E , typename OffsetKind >
typedef E  rti::flat::SequenceIterator< E, OffsetKind >::value_type
```

The element type.

8.340.2.3 reference

```
template<typename E , typename OffsetKind >
typedef  value_type  rti::flat::SequenceIterator< E, OffsetKind >::reference
```

The reference type is the same as the value type, an Offset.

8.340.2.4 pointer

```
template<typename E , typename OffsetKind >
typedef  value_type  rti::flat::SequenceIterator< E, OffsetKind >::pointer
```

The pointer type is the same as the value type, an Offset.

8.340.2.5 difference_type

```
template<typename E , typename OffsetKind >
typedef std::ptrdiff_t  rti::flat::SequenceIterator< E, OffsetKind >::difference_type
```

The difference type.

8.340.3 Constructor & Destructor Documentation

8.340.3.1 SequenceIterator()

```
template<typename E , typename OffsetKind >
rti::flat::SequenceIterator< E, OffsetKind >::SequenceIterator ( )  [inline]
```

Constructs an invalid iterator.

8.340.4 Member Function Documentation

8.340.4.1 is_null()

```
template<typename E , typename OffsetKind >
bool  rti::flat::SequenceIterator< E, OffsetKind >::is_null ( ) const  [inline]
```

Returns whether the iterator is invalid.

8.340.4.2 operator*()

```
template<typename E , typename OffsetKind >
value_type  rti::flat::SequenceIterator< E, OffsetKind >::operator* ( ) const  [inline]
```

Returns the Offset of the current element.

Returns

The Offset to the current element. Note that this function returns by value, not by reference. This Offset may be **null** (p. 1876) if the iterator has surpassed the length of the collection. See **Offset Error Management** (p. 416).

8.340.4.3 operator->()

```
template<typename E , typename OffsetKind >
value_type  rti::flat::SequenceIterator< E, OffsetKind >::operator-> ( ) const  [inline]
```

Returns the Offset of the current element.

8.340.4.4 advance()

```
template<typename E , typename OffsetKind >
bool  rti::flat::SequenceIterator< E, OffsetKind >::advance ( )  [inline]
```

Advances to the next element, reporting any errors by returning false.

Unlike operator++, which throws an exception in case of error, **advance()** (p. 2333) returns false

Returns

True if the function succeeds, or false if there was an error

8.340.4.5 operator++() [1/2]

```
template<typename E , typename OffsetKind >
SequenceIterator &  rti::flat::SequenceIterator< E, OffsetKind >::operator++ ( )  [inline]
```

Advances to the next element.

8.340.4.6 operator++() [2/2]

```
template<typename E , typename OffsetKind >
SequenceIterator  rti::flat::SequenceIterator< E, OffsetKind >::operator++ (
            int  )  [inline]
```

Advances to the next element.

8.340.5 Friends And Related Function Documentation

8.340.5.1 operator<

```
template<typename E , typename OffsetKind >
bool operator< (
            const  SequenceIterator< E, OffsetKind > & s1,
            const  SequenceIterator< E, OffsetKind > & s2 )  [friend]
```

Compares two iterators.

8.340.5.2 operator>

```
template<typename E , typename OffsetKind >
bool operator> (
            const  SequenceIterator< E, OffsetKind > & s1,
            const  SequenceIterator< E, OffsetKind > & s2 )  [friend]
```

Compares two iterators.

8.340.5.3 operator<=

```
template<typename E , typename OffsetKind >
bool operator<= (
            const  SequenceIterator< E, OffsetKind > & s1,
            const  SequenceIterator< E, OffsetKind > & s2 )  [friend]
```

Compares two iterators.

8.340.5.4 operator>=

```
template<typename E , typename OffsetKind >
bool operator>= (
            const  SequenceIterator< E, OffsetKind > & s1,
            const  SequenceIterator< E, OffsetKind > & s2 )  [friend]
```

Compares two iterators.

8.340.5.5 operator==

```
template<typename E , typename OffsetKind >
bool operator== (
            const  SequenceIterator< E, OffsetKind > & s1,
            const  SequenceIterator< E, OffsetKind > & s2 )  [friend]
```

Compares two iterators.

8.340.5.6 operator"!=

```
template<typename E , typename OffsetKind >
bool operator!= (
            const  SequenceIterator< E, OffsetKind > & s1,
            const  SequenceIterator< E, OffsetKind > & s2 )  [friend]
```

Compares two iterators.

8.341 rti::core::SequenceNumber Class Reference

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* A class representing the DDS 64-bit Sequence Number

```
#include <rti/core/SequenceNumber.hpp>
```

Public Member Functions

- **SequenceNumber** ()

  *Create a default **SequenceNumber** (p. 2335), equal to **unknown()** (p. 2338)*
- **SequenceNumber** (int32_t high_value, uint32_t low_value)

  *Create a **SequenceNumber** (p. 2335) with the provided high and low values.*
- **SequenceNumber** (int64_t the_value)

  *Creates a **SequenceNumber** (p. 2335) from an int64_t.*
- int32_t **high** () const

  Get the value of the most significant part of the sequence number.
- **SequenceNumber** & **high** (int32_t the_value)

  Set the value of the most significant part of the sequence number.
- uint32_t **low** () const

  Get the value of the most significant part of the sequence number.
- **SequenceNumber** & **low** (uint32_t the_value)

  Set the value of the least significant part of the sequence number.
- long long **value** () const

  Get the value of the sequence number.
- void **value** (long long value)

  Set the value of the sequence number.
- **SequenceNumber** **operator+** (const **SequenceNumber** &other) const

  Add two SequenceNumbers.
- **SequenceNumber** **operator-** (const **SequenceNumber** &other) const

  *Subtract one **SequenceNumber** (p. 2335) from another.*
- **SequenceNumber** & **operator+=** (const **SequenceNumber** &other)

  *Compound assignment operator, assigning the result of the addition to the current **SequenceNumber** (p. 2335) object.*
- **SequenceNumber** & **operator-=** (const **SequenceNumber** &other)

  *Compound assignment operator, assigning the result of the subtraction to the current **SequenceNumber** (p. 2335) object.*
- **SequenceNumber** **operator++** ()

  *Pre-increment the value of the **SequenceNumber** (p. 2335) by 1.*
- **SequenceNumber** **operator++** (int)

  *Post-increment the value of the **SequenceNumber** (p. 2335) by 1.*
- **SequenceNumber** **operator--** ()

  *Pre-decrement the value of the **SequenceNumber** (p. 2335) by 1.*
- **SequenceNumber** **operator--** (int)

  *Post-decrement the value of the **SequenceNumber** (p. 2335) by 1.*
- bool **operator<** (const **SequenceNumber** &other) const

  Compare two SequenceNumbers for a less-than relationship.
- bool **operator<=** (const **SequenceNumber** &other) const

  Compare two SequenceNumbers for a less-than-or-equal relationship.
- bool **operator>** (const **SequenceNumber** &other) const

  Compare two SequenceNumbers for a greater-than relationship.
- bool **operator>=** (const **SequenceNumber** &other) const

  Compare two SequenceNumbers for a greater-than-or-equal relationship.

Static Public Member Functions

- static **SequenceNumber zero** ()

  *Create a **SequenceNumber** (p. 2335) representing 0.*
- static **SequenceNumber unknown** ()

  *Create a **SequenceNumber** (p. 2335) representing the unknown **SequenceNumber** (p. 2335) value.*
- static **SequenceNumber maximum** ()

  *Create a **SequenceNumber** (p. 2335) representing the highest, most positive value for the sequence number.*
- static **SequenceNumber automatic** ()

  *Create a **SequenceNumber** (p. 2335) that will cause the value to be internally determined by RTI Connext.*

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator**<< (std::ostream &out, const **SequenceNumber** &sn)

  *Prints a **SequenceNumber** (p. 2335) to an output stream.*

8.341.1 Detailed Description

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* A class representing the DDS 64-bit Sequence Number

8.341.2 Constructor & Destructor Documentation

8.341.2.1 SequenceNumber() [1/3]

```
rti::core::SequenceNumber::SequenceNumber ( )  [inline]
```

Create a default **SequenceNumber** (p. 2335), equal to **unknown()** (p. 2338)

8.341.2.2 SequenceNumber() [2/3]

```
rti::core::SequenceNumber::SequenceNumber (
            int32_t high_value,
            uint32_t low_value )  [inline]
```

Create a **SequenceNumber** (p. 2335) with the provided high and low values.

Parameters

high_value	The value for the most significant part of the sequence number.
low_value	The value for the least significant part of the sequence number.

8.341.2.3 SequenceNumber() [3/3]

```
rti::core::SequenceNumber::SequenceNumber (
            int64_t the_value )  [inline]
```

Creates a **SequenceNumber** (p. 2335) from an int64_t.

8.341.3 Member Function Documentation

8.341.3.1 zero()

```
static  SequenceNumber rti::core::SequenceNumber::zero ( )  [inline], [static]
```

Create a **SequenceNumber** (p. 2335) representing 0.

8.341.3.2 unknown()

```
static  SequenceNumber rti::core::SequenceNumber::unknown ( )  [inline], [static]
```

Create a **SequenceNumber** (p. 2335) representing the unknown **SequenceNumber** (p. 2335) value.

8.341.3.3 maximum()

```
static  SequenceNumber rti::core::SequenceNumber::maximum ( )  [inline], [static]
```

Create a **SequenceNumber** (p. 2335) representing the highest, most positive value for the sequence number.

8.341.3.4 automatic()

```
static  SequenceNumber rti::core::SequenceNumber::automatic ( )  [inline], [static]
```

Create a **SequenceNumber** (p. 2335) that will cause the value to be internally determined by RTI Connext.

See also

rti::core::WriteParams::replace_automatic_values()

8.341.3.5 high() [1/2]

```
int32_t rti::core::SequenceNumber::high ( ) const  [inline]
```

Get the value of the most significant part of the sequence number.

8.341.3.6 high() [2/2]

```
SequenceNumber & rti::core::SequenceNumber::high (
            int32_t the_value )  [inline]
```

Set the value of the most significant part of the sequence number.

8.341.3.7 low() [1/2]

```
uint32_t rti::core::SequenceNumber::low ( ) const  [inline]
```

Get the value of the most significant part of the sequence number.

8.341.3.8 low() [2/2]

```
SequenceNumber & rti::core::SequenceNumber::low (
            uint32_t the_value )  [inline]
```

Set the value of the least significant part of the sequence number.

8.341.3.9 value() [1/2]

```
long long rti::core::SequenceNumber::value ( ) const
```

Get the value of the sequence number.

8.341.3.10 value() [2/2]

```
void rti::core::SequenceNumber::value (
            long long value )
```

Set the value of the sequence number.

8.341.3.11 operator+()

```
SequenceNumber rti::core::SequenceNumber::operator+ (
            const  SequenceNumber & other ) const
```

Add two SequenceNumbers.

8.341.3.12 operator-()

```
SequenceNumber rti::core::SequenceNumber::operator- (
            const  SequenceNumber & other ) const
```

Subtract one **SequenceNumber** (p. 2335) from another.

8.341.3.13 operator+=()

```
SequenceNumber & rti::core::SequenceNumber::operator+= (
            const  SequenceNumber & other )
```

Compound assignment operator, assigning the result of the addition to the current **SequenceNumber** (p. 2335) object.

8.341.3.14 operator-=()

```
SequenceNumber & rti::core::SequenceNumber::operator-= (
            const  SequenceNumber & other )
```

Compound assignment operator, assigning the result of the subtraction to the current **SequenceNumber** (p. 2335) object.

8.341.3.15 operator++() [1/2]

```
SequenceNumber rti::core::SequenceNumber::operator++ ( )
```

Pre-increment the value of the **SequenceNumber** (p. 2335) by 1.

8.341.3.16 operator++() [2/2]

```
SequenceNumber rti::core::SequenceNumber::operator++ (
            int  )
```

Post-increment the value of the **SequenceNumber** (p. 2335) by 1.

8.341.3.17 operator--() [1/2]

```
SequenceNumber rti::core::SequenceNumber::operator-- ( )
```

Pre-decrement the value of the **SequenceNumber** (p. 2335) by 1.

8.341.3.18 operator--() [2/2]

```
SequenceNumber rti::core::SequenceNumber::operator-- (
            int  )
```

Post-decrement the value of the **SequenceNumber** (p. 2335) by 1.

8.341.3.19 operator<()

```
bool rti::core::SequenceNumber::operator< (
            const  SequenceNumber & other ) const
```

Compare two SequenceNumbers for a less-than relationship.

8.341.3.20 operator<=()

```
bool rti::core::SequenceNumber::operator<= (
            const  SequenceNumber & other ) const
```

Compare two SequenceNumbers for a less-than-or-equal relationship.

8.341.3.21 operator>()

```
bool rti::core::SequenceNumber::operator> (
            const  SequenceNumber & other ) const
```

Compare two SequenceNumbers for a greater-than relationship.

8.341.3.22 operator>=()

```
bool rti::core::SequenceNumber::operator>= (
            const  SequenceNumber & other ) const
```

Compare two SequenceNumbers for a greater-than-or-equal relationship.

8.341.4 Friends And Related Function Documentation

8.341.4.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const  SequenceNumber & sn )  [related]
```

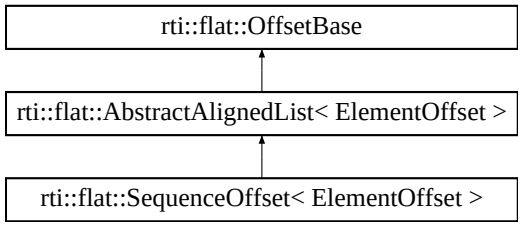
Prints a **SequenceNumber** (p. 2335) to an output stream.

8.342 rti::flat::SequenceOffset< ElementOffset > Class Template Reference

Offset to a sequence of non-primitive elements.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::SequenceOffset< ElementOffset >:



Public Member Functions

- ElementOffset **get_element** (unsigned int i)

  Gets the Offset to an element.
- unsigned int **element_count** () const

  The number of elements.

Additional Inherited Members

8.342.1 Detailed Description

template<typename ElementOffset>
class rti::flat::SequenceOffset< ElementOffset >

Offset to a sequence of non-primitive elements.

Template Parameters

ElementOffset	The Offset type, for example **MyFlatMutableOffset** (p. 1770), **MyFlatFinalOffset** (p. 1759), or **StringOffset** (p. 2394).

Represents an Offset to a sequence member and allows getting an Offset to each of its elements.

A **SequenceOffset** (p. 2343) may meet the requirements to be cast to an array of the equivalent plain C++ element type (see **rti::flat::plain_cast** (p. 417)), if the `ElementOffset` is a final type.

8.342.2 Member Function Documentation

8.342.2.1 get_element()

```
template<typename ElementOffset >
ElementOffset  rti::flat::SequenceOffset< ElementOffset >::get_element (
            unsigned int i )  [inline]
```

Gets the Offset to an element.

Parameters

i	The zero-based index to the element

Returns

The Offset to the element in the i-th position

8.342.2.2 element_count()

```
template<typename ElementOffset >
unsigned int  rti::flat::SequenceOffset< ElementOffset >::element_count ( ) const  [inline]
```

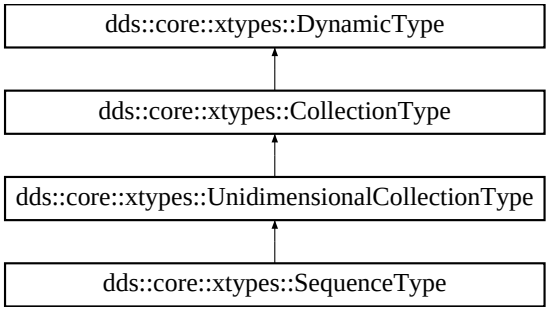
The number of elements.

8.343 dds::core::xtypes::SequenceType Class Reference

<<***value-type***>> *(p. 350)* Represents an IDL `sequence` type.

`#include <dds/core/xtypes/CollectionTypes.hpp>`

Inheritance diagram for dds::core::xtypes::SequenceType:



Public Member Functions

- **SequenceType** (const **dds::core::xtypes::DynamicType** &type)

  Creates an unbounded collection with an element type.
- **SequenceType** (const **dds::core::xtypes::DynamicType** &type, uint32_t **bounds**)

  Creates a bounded collection with an element type.
- **SequenceType** (**dds::core::xtypes::DynamicType** &&type)

  <<***C++11***>> *(*p. *353) Creates an unbounded collection with an element type*
- **SequenceType** (**dds::core::xtypes::DynamicType** &&type, uint32_t the_bounds)

  <<***C++11***>> *(*p. *353) Creates a bounded collection with an element type*

Additional Inherited Members

8.343.1 Detailed Description

<<***value-type***>> *(*p. *350)* Represents an IDL `sequence` type.

8.343.2 Constructor & Destructor Documentation

8.343.2.1 SequenceType() [1/4]

```
dds::core::xtypes::SequenceType::SequenceType (
            const  dds::core::xtypes::DynamicType & type )  [explicit]
```

Creates an unbounded collection with an element type.

Parameters

type	The element type

8.343.2.2 SequenceType() [2/4]

```
dds::core::xtypes::SequenceType::SequenceType (
            const  dds::core::xtypes::DynamicType & type,
            uint32_t bounds )
```

Creates a bounded collection with an element type.

Parameters

type	The element type
bounds	The maximum length

8.343.2.3 SequenceType() [3/4]

```
dds::core::xtypes::SequenceType::SequenceType (
            dds::core::xtypes::DynamicType && type )  [inline], [explicit]
```

<<***C++11***>> *(p. 353)* Creates an unbounded collection with an element type

Parameters

type	The element type

8.343.2.4 SequenceType() [4/4]

```
dds::core::xtypes::SequenceType::SequenceType (
            dds::core::xtypes::DynamicType && type,
           uint32_t the_bounds )  [inline]
```

<<***C++11***>> *(p. 353)* Creates a bounded collection with an element type

Parameters

type	The element type
the_bounds	The maximum length

8.344 dds::rpc::Server Class Reference

<<***reference-type***>> *(p. 351)* Provides the execution environment for one or more **ServiceEndpoint** (p. 2354).

`#include <dds/rpc/Server.hpp>`

Public Member Functions

- **Server** ()

*Creates a new server with default **ServerParams** (p. 2348).*

- **Server** (const **ServerParams** &params)

  *Creates a new server with the specified **ServerParams** (p. 2348).*
- void **run** ()

  Holds the execution of the current thread.
- void **run** (const **dds::core::Duration** &max_wait)

  Holds the execution of the current thread for the specified amount of time.
- void **close** ()

  Forces the destruction of this entity.

8.344.1 Detailed Description

<<***reference-type***>> *(p. 351)* Provides the execution environment for one or more **ServiceEndpoint** (p. 2354).

8.344.2 Constructor & Destructor Documentation

8.344.2.1 Server() [1/2]

```
dds::rpc::Server::Server ( )  [inline]
```

Creates a new server with default **ServerParams** (p. 2348).

8.344.2.2 Server() [2/2]

```
dds::rpc::Server::Server (
            const  ServerParams & params )  [inline], [explicit]
```

Creates a new server with the specified **ServerParams** (p. 2348).

8.344.3 Member Function Documentation

8.344.3.1 run() [1/2]

```
void dds::rpc::Server::run ( )  [inline]
```

Holds the execution of the current thread.

Calling this function is optional. A **Server** (p. 2346) manages a thread pool that runs as soon as a **ServiceEndpoint** (p. 2354) is attached.

8.344.3.2 run() [2/2]

```
void dds::rpc::Server::run (
            const  dds::core::Duration & max_wait )  [inline]
```

Holds the execution of the current thread for the specified amount of time.

Calling this function is optional. A **Server** (p. 2346) manages a thread pool that runs as soon as a **ServiceEndpoint** (p. 2354) is attached.

Parameters

max_wait	The time to sleep

8.344.3.3 close()

```
void dds::rpc::Server::close ( )  [inline]
```

Forces the destruction of this entity.

If **run()** (p. 2347) is currently blocked on another thread, **close()** (p. 2348) unblocks it.

Any operation after **close()** (p. 2348) throws **dds::core::AlreadyClosedError** (p. 806)

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

See also

Reference types (p. 351)

8.345 dds::rpc::ServerParams Class Reference

<<***value-type***>> *(p. 350)* The parameters used to configure a **Server** (p. 2346)

```
#include <dds/rpc/ServerParams.hpp>
```

Public Member Functions

- void **thread_pool_size** (int32_t size)

  *Configures the number of threads of a **Server** (p. 2346) thread pool.*
- int32_t **thread_pool_size** () const

  Returns the thread pool size.
- **rti::core::cond::AsyncWaitSetProperty** & **async_waitset_property** ()

  Allows fine-tuning the internal AsyncWaitSet used to process function calls.
- const **rti::core::cond::AsyncWaitSetProperty** & **async_waitset_property** () const

  Returns the AsyncWaitSetProperty by const reference.

8.345.1 Detailed Description

<<***value-type***>> *(p. 350)* The parameters used to configure a **Server** (p. 2346)

8.345.2 Member Function Documentation

8.345.2.1 thread_pool_size() [1/2]

```
void thread_pool_size (
            int32_t size )
```

Configures the number of threads of a **Server** (p. 2346) thread pool.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

By default a server uses only one thread, processing all function calls sequentially. Incresing the pool size provides more parallelism.

8.345.2.2 thread_pool_size() [2/2]

```
int32_t thread_pool_size ( ) const
```

Returns the thread pool size.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.345.2.3 async_waitset_property() [1/2]

```
rti::core::cond::AsyncWaitSetProperty & async_waitset_property ( )
```

Allows fine-tuning the internal AsyncWaitSet used to process function calls.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.345.2.4 async_waitset_property() [2/2]

```
const rti::core::cond::AsyncWaitSetProperty & async_waitset_property ( ) const
```

Returns the AsyncWaitSetProperty by const reference.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.346 rti::core::policy::Service Class Reference

<<***extension***>> *(p. 116)* Indicates if an Entity is associated with a service or tool and if so, which one.

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **Service** ()
  Creates the default policy (no service)
- **Service** (**rti::core::policy::ServiceKind** the_kind)
  Creates an instance with the specified service kind.
- **Service** & **kind** (const **rti::core::policy::ServiceKind** the_kind)
  Sets the service kind.
- **rti::core::policy::ServiceKind kind** () const
  Gets the service kind.

Static Public Member Functions

- static **Service NoService** ()

  *Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::NO_SERVICE** (p. 2358).*
- static **Service PersistenceService** ()

  *Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::PERSISTENCE** (p. 2358).*
- static **Service QueuingService** ()

  *Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::QUEUING** (p. 2358).*
- static **Service RoutingService** ()

  *Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::ROUTING** (p. 2358).*
- static **Service RecordingService** ()

  *Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::RECORDING** (p. 2358).*
- static **Service ReplayService** ()

  *Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::REPLAY** (p. 2358).*
- static **Service DatabaseIntegrationService** ()

  *Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::DATABASE_INTEGRATION** (p. 2358).*
- static **Service WebIntegrationService** ()

  *Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::WEB_INTEGRATION** (p. 2358).*
- static **Service ObservabilityCollectorService** ()

  *Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::OBSERVABILITY_COLLECTOR** (p. 2358).*
- static **Service AdminConsoleService** ()

  *Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::ADMIN_CONSOLE** (p. 2358).*

8.346.1 Detailed Description

<<***extension***>> *(p. 116)* Indicates if an Entity is associated with a service or tool and if so, which one.

This QoS policy is intended to be used by RTI infrastructure services and RTI tools.

User applications should not modify its value.

Entity:

dds::domain::DomainParticipant (p. 1326), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Properties:

RxO (p. 514) = NO
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

8.346.2 Constructor & Destructor Documentation

8.346.2.1 Service() [1/2]

```
rti::core::policy::Service::Service ( )  [inline]
```

Creates the default policy (no service)

8.346.2.2 Service() [2/2]

```
rti::core::policy::Service::Service (
            rti::core::policy::ServiceKind the_kind )  [inline]
```

Creates an instance with the specified service kind.

8.346.3 Member Function Documentation

8.346.3.1 NoService()

```
static  Service rti::core::policy::Service::NoService ( )  [inline], [static]
```

Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::NO_SERVICE** (p. 2358).

8.346.3.2 PersistenceService()

```
static  Service rti::core::policy::Service::PersistenceService ( )  [inline], [static]
```

Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::PERSISTENCE** (p. 2358).

8.346.3.3 QueuingService()

```
static  Service rti::core::policy::Service::QueuingService ( )  [inline], [static]
```

Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::QUEUING** (p. 2358).

8.346.3.4 RoutingService()

```
static Service rti::core::policy::Service::RoutingService ( ) [inline], [static]
```

Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::ROUTING** (p. 2358).

8.346.3.5 RecordingService()

```
static Service rti::core::policy::Service::RecordingService ( ) [inline], [static]
```

Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::RECORDING** (p. 2358).

8.346.3.6 ReplayService()

```
static Service rti::core::policy::Service::ReplayService ( ) [inline], [static]
```

Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::REPLAY** (p. 2358).

8.346.3.7 DatabaseIntegrationService()

```
static Service rti::core::policy::Service::DatabaseIntegrationService ( ) [inline], [static]
```

Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::DATABASE_INTEGRATION** (p. 2358).

8.346.3.8 WebIntegrationService()

```
static Service rti::core::policy::Service::WebIntegrationService ( ) [inline], [static]
```

Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::WEB_INTEGRATION** (p. 2358).

8.346.3.9 ObservabilityCollectorService()

```
static Service rti::core::policy::Service::ObservabilityCollectorService ( ) [inline], [static]
```

Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::OBSERVABILITY_COLLECTOR** (p. 2358).

8.346.3.10 AdminConsoleService()

```
static  Service rti::core::policy::Service::AdminConsoleService ( )  [inline], [static]
```

Returns a **Service** (p. 2350) with **rti::core::policy::ServiceKind_def::ADMIN_CONSOLE** (p. 2358).

8.346.3.11 kind() [1/2]

```
Service & rti::core::policy::Service::kind (
            const  rti::core::policy::ServiceKind the_kind )
```

Sets the service kind.

8.346.3.12 kind() [2/2]

```
rti::core::policy::ServiceKind rti::core::policy::Service::kind ( ) const
```

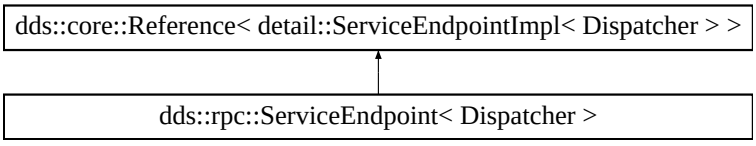
Gets the service kind.

8.347 dds::rpc::ServiceEndpoint< Dispatcher > Class Template Reference

Manages the DDS entities required to receive function calls and send the return values.

```
#include <ServiceEndpoint.hpp>
```

Inheritance diagram for dds::rpc::ServiceEndpoint< Dispatcher >:



Public Types

- using **InterfaceType** = typename Dispatcher::InterfaceType
  *The interface type, such as **rpc_example::RobotControl** (p. 2213).*
- using **RequestType** = typename Dispatcher::RequestType
  The internal type used to receive requests (function calls)
- using **ReplyType** = typename Dispatcher::ReplyType
  The internal type used to send replies (the return values of the function calls)

Public Member Functions

- **ServiceEndpoint** (std::shared_ptr< **InterfaceType** > service_impl, **dds::rpc::Server** server, const **ServiceParams** &params)

  Construct a new Service Endpoint object.
- void **close** ()

  Destroys the underlying resources.
- **dds::sub::DataReader**< **RequestType** > **request_datareader** ()

  Accesses the underlying DataReader that receives the requests (function calls)
- **dds::pub::DataWriter**< **ReplyType** > **reply_datawriter** ()

  Accesses the underlying DataWriter that sends the replies (return values from the function calls)

8.347.1 Detailed Description

template<typename Dispatcher>
class dds::rpc::ServiceEndpoint< Dispatcher >

Manages the DDS entities required to receive function calls and send the return values.

Template Parameters

Dispatcher	Internal type generated by `rtiddsgen`

Applications shouldn't use this class directly. `rtiddsgen` will generate the proper instantiation of this class, such as rpc_example::RobotControlService

A **ServiceEndpoint** (p. 2354) requires a an implementation of an IDL service interface, such as **rpc_example::RobotControl** (p. 2213).

A **ServiceEndpoint** (p. 2354) is attached to a **Server** (p. 2346) in order to start receiving and processing remote function calls.

8.347.2 Member Typedef Documentation

8.347.2.1 InterfaceType

```
template<typename Dispatcher >
using dds::rpc::ServiceEndpoint< Dispatcher >::InterfaceType = typename Dispatcher::Interface↩
Type
```

The interface type, such as **rpc_example::RobotControl** (p. 2213).

8.347.2.2 RequestType

```
template<typename Dispatcher >
using  dds::rpc::ServiceEndpoint< Dispatcher >::RequestType = typename Dispatcher::RequestType
```

The internal type used to receive requests (function calls)

8.347.2.3 ReplyType

```
template<typename Dispatcher >
using  dds::rpc::ServiceEndpoint< Dispatcher >::ReplyType = typename Dispatcher::ReplyType
```

The internal type used to send replies (the return values of the function calls)

8.347.3 Constructor & Destructor Documentation

8.347.3.1 ServiceEndpoint()

```
template<typename Dispatcher >
dds::rpc::ServiceEndpoint< Dispatcher >::ServiceEndpoint (
            std::shared_ptr<  InterfaceType > service_impl,
             dds::rpc::Server server,
            const  ServiceParams & params )  [inline]
```

Construct a new Service Endpoint object.

Parameters

service_impl	The implementation of the DDS service, such as **rpc_example::RobotControl** (p. 2213).
server	The server where to run this service endpoint.
params	The parameters used to configure this service endpoint

8.347.4 Member Function Documentation

8.347.4.1 close()

```
template<typename Dispatcher >
void  dds::rpc::ServiceEndpoint< Dispatcher >::close ( )  [inline]
```

Destroys the underlying resources.

Any operation after **close()** (p. 2356) throws **dds::core::AlreadyClosedError** (p. 806)

8.347.4.2 request_datareader()

```
template<typename Dispatcher >
dds::sub::DataReader< RequestType > dds::rpc::ServiceEndpoint< Dispatcher >::request_datareader
( ) [inline]
```

Accesses the underlying DataReader that receives the requests (function calls)

Returns

The DataReader

8.347.4.3 reply_datawriter()

```
template<typename Dispatcher >
dds::pub::DataWriter< ReplyType > dds::rpc::ServiceEndpoint< Dispatcher >::reply_datawriter (
) [inline]
```

Accesses the underlying DataWriter that sends the replies (return values from the function calls)

Returns

The DataWriter

8.348 rti::core::policy::ServiceKind_def Struct Reference

<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) ServiceKind

```
#include <PolicyKind.hpp>
```

Public Types

- enum **type** {
  NO_SERVICE ,
  PERSISTENCE ,
  QUEUING ,
  ROUTING ,
  RECORDING ,
  REPLAY ,
  DATABASE_INTEGRATION ,
  WEB_INTEGRATION ,
  OBSERVABILITY_COLLECTOR ,
  ADMIN_CONSOLE }

  The underlying `enum` *type.*

8.348.1 Detailed Description

<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) ServiceKind

8.348.2 Member Enumeration Documentation

8.348.2.1 type

`enum` **`rti::core::policy::ServiceKind_def::type`**

The underlying `enum` type.

Enumerator

NO_SERVICE	There is no service associated to the Entity.
PERSISTENCE	The Entity is associated to RTI Persistence **Service** (p. 2350).
QUEUING	The Entity is associated to RTI Queuing **Service** (p. 2350).
ROUTING	The Entity is associated to RTI Routing **Service** (p. 2350).
RECORDING	The Entity is associated to RTI Recording **Service** (p. 2350).
REPLAY	The Entity is associated to RTI Replay **Service** (p. 2350).
DATABASE_INTEGRATION	The Entity is associated to RTI **Database** (p. 969) Integration **Service** (p. 2350).
WEB_INTEGRATION	The Entity is associated to RTI Web Integration **Service** (p. 2350).
OBSERVABILITY_COLLECTOR	The Entity is associated to RTI Observability Collector **Service** (p. 2350).
ADMIN_CONSOLE	The Entity is associated to RTI Admin Console.

8.349 rti::topic::ServiceRequest Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* A request coming from one of the built-in services

`#include <dds/topic/BuiltinTopic.hpp>`

Public Member Functions

- **rti::core::ServiceRequestId service_id** () const

  Get the service id of the request.
- const **rti::core::Guid** & **instance_id** () const

  Get the instance id of the request.
- const **dds::core::vector**< uint8_t > & **request_body** () const

  Get the request body of the request.

8.349.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* A request coming from one of the built-in services

Data associated with the built-in topic **rti::topic::service_request_topic_name()** (p. 568). It contains service-specific information.

See also

rti::topic::service_request_topic_name() (p. 568)

8.349.2 Member Function Documentation

8.349.2.1 service_id()

```
rti::core::ServiceRequestId rti::topic::ServiceRequest::service_id ( ) const  [inline]
```

Get the service id of the request.

Returns

The service id of the request

8.349.2.2 instance_id()

```
const  rti::core::Guid & rti::topic::ServiceRequest::instance_id ( ) const  [inline]
```

Get the instance id of the request.

Returns

The instance id of the request

8.349.2.3 request_body()

```
const  dds::core::vector< uint8_t > & rti::topic::ServiceRequest::request_body ( ) const  [inline]
```

Get the request body of the request.

Returns

The request body of the request

8.350 rti::core::status::ServiceRequestAcceptedStatus Class Reference

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::service_request_accepted()** (p. 2388)

`#include <Status.hpp>`

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **EventCount32 total_count** () const

  *The total cumulative number of ServiceRequests that have been accepted by a **dds::pub::DataWriter** (p. 1138).*
- **EventCount32 current_count** () const

  *The current number of ServiceRequests that have been accepted by this **dds::pub::DataWriter** (p. 1138).*
- **dds::core::InstanceHandle last_request_handle** () const

  *A handle to the last **rti::topic::ServiceRequest** (p. 2358) that caused the **dds::pub::DataWriter** (p. 1138)'s status to change.*
- **rti::core::ServiceRequestId service_id** () const

  ID of the service to which the accepted Request belongs.

8.350.1 Detailed Description

<<***extension***>> *(p. 354)* Information about the status **dds::core::status::StatusMask::service_request_accepted()** (p. 2388)

Currently, the only service that causes the **ServiceRequestAcceptedStatus** (p. 2360) to be triggered is the **rti::sub::TopicQuery** (p. 2516) service. A **rti::topic::ServiceRequest** (p. 2358) is accepted when a **dds::pub::DataWriter** (p. 1138) matches with a **dds::sub::DataReader** (p. 976) that has created a **rti::sub::TopicQuery** (p. 2516).

This status is also changed (and the listener, if any, called) when a ServiceRequest has been cancelled, or deleted. This will happen when a **dds::sub::DataReader** (p. 976) deletes a TopicQuery using **rti::sub::TopicQuery::close()** (p. 2518).

8.350.2 Member Function Documentation

8.350.2.1 total_count()

```
EventCount32 rti::core::status::ServiceRequestAcceptedStatus::total_count ( ) const  [inline]
```

The total cumulative number of ServiceRequests that have been accepted by a **dds::pub::DataWriter** (p. 1138).

This number increases whenever a new request is accepted. It does not change when a request is cancelled.

8.350.2.2 current_count()

```
EventCount32 rti::core::status::ServiceRequestAcceptedStatus::current_count ( ) const  [inline]
```

The current number of ServiceRequests that have been accepted by this **dds::pub::DataWriter** (p. 1138).

This number increases when a new request is accepted and decreases when an existing request is cancelled.

8.350.2.3 last_request_handle()

```
dds::core::InstanceHandle rti::core::status::ServiceRequestAcceptedStatus::last_request_handle (
) const  [inline]
```

A handle to the last **rti::topic::ServiceRequest** (p. 2358) that caused the **dds::pub::DataWriter** (p. 1138)'s status to change.

8.350.2.4 service_id()

```
rti::core::ServiceRequestId rti::core::status::ServiceRequestAcceptedStatus::service_id ( ) const
[inline]
```

ID of the service to which the accepted Request belongs.

See also

rti::core::ServiceRequestId_def::TOPIC_QUERY (p. 2362)

8.351 rti::core::ServiceRequestId_def Struct Reference

The definition of the rti::core::safe_enum ServiceRequestId.

```
#include <rti/core/ServiceRequestId.hpp>
```

Public Types

- enum **type** {
  UNKNOWN = PRES_SERVICE_REQUEST_ID_UNKNOWN ,
  TOPIC_QUERY = PRES_SERVICE_REQUEST_ID_TOPIC_QUERY ,
  LOCATOR_REACHABILITY = PRES_SERVICE_REQUEST_ID_LOCATOR_REACHABILITY ,
  INSTANCE_STATE = PRES_SERVICE_REQUEST_ID_INSTANCE_STATE ,
  MONITORING_LIBRARY_COMMAND ,
  MONITORING_LIBRARY_REPLY }

  The underlying `enum` *type.*

8.351.1 Detailed Description

The definition of the rti::core::safe_enum ServiceRequestId.

Use ServiceRequestId.

8.351.2 Member Enumeration Documentation

8.351.2.1 type

`enum` **rti::core::ServiceRequestId_def::type**

The underlying `enum` type.

Enumerator

UNKNOWN	An unknown service.
TOPIC_QUERY	The topic query service.
LOCATOR_REACHABILITY	The locator reachability service.
INSTANCE_STATE	The instance state service.
MONITORING_LIBRARY_COMMAND	The Monitoring Library 2.0 command service.
MONITORING_LIBRARY_REPLY	The Monitoring Library 2.0 reply service.

8.352 dds::sub::SharedSamples< T, DELEGATE > Class Template Reference

<<***reference-type***>> *(*p. *351)* A sharable and container-safe version of **LoanedSamples** (p. 1663).

`#include <dds/sub/SharedSamples.hpp>`

Public Member Functions

- **SharedSamples** (**dds::sub::LoanedSamples**< T > &ls)

  Constructs and instance of ***SharedSamples*** *(*p. *2362) removing the ownership of the loan from the* ***LoanedSamples*** *(*p. *1663).*
- const_iterator **begin** () const

  Same as ***LoanedSamples::begin()*** *(*p. *1667)*
- const_iterator **end** () const

  Same as ***LoanedSamples::end()*** *(*p. *1668)*
- DELEGATE< T >::value_type **operator[ ]** (size_t index) const

  Same as ***LoanedSamples::operator[ ](size_t)*** *(*p. *1666)*
- uint32_t **length** () const

  Returns the number of samples.

Related Functions

(Note that these are not member functions.)

- template<typename T >
  void **unpack** (const **dds::sub::SharedSamples**< T > &samples, std::vector< std::shared_ptr< const T > > &sample_vector)

  <<***extension***>> *(p. 354)* <<***C++11***>> *(p. 353) Unpacks a* ***SharedSamples*** *(p. 2362) collection into individual shared_ptr's in a vector*
- template<typename T >
  std::vector< std::shared_ptr< const T > > **unpack** (const **dds::sub::SharedSamples**< T > &samples)

  <<***extension***>> *(p. 354)* <<***C++11***>> *(p. 353) Unpacks a* ***SharedSamples*** *(p. 2362) collection into individual shared_ptr's in a vector*
- template<typename T >
  std::vector< std::shared_ptr< const T > > **unpack** (**dds::sub::LoanedSamples**< T > &&samples)

  <<***extension***>> *(p. 354)* <<***C++11***>> *(p. 353) Unpacks a* ***LoanedSamples*** *(p. 1663) collection into individual shared_ptr's in a vector*

8.352.1 Detailed Description

template<typename T, template< typename Q > class DELEGATE = detail::SharedSamples>
class dds::sub::SharedSamples< T, DELEGATE >

<<***reference-type***>> *(p. 351)* A sharable and container-safe version of **LoanedSamples** (p. 1663).

A **SharedSamples** (p. 2362) instance is constructed from a **LoanedSamples** (p. 1663) instance, removing the ownership of the loan from the **LoanedSamples** (p. 1663).

The destruction of the **LoanedSamples** (p. 1663) object or the explicit invocation of its method return_loan will have no effect on loaned data.

Constructing a **SharedSamples** (p. 2362) from another **SharedSamples** (p. 2362), creates a new reference to the same loan. Loaned data will be returned automatically to the **DataReader** (p. 976) once the reference count reaches zero.

See also

LoanedSamples (p. 1663)

8.352.2 Constructor & Destructor Documentation

8.352.2.1 SharedSamples()

```
template<typename T , template< typename Q > class DELEGATE = detail::SharedSamples>
dds::sub::SharedSamples< T, DELEGATE >::SharedSamples (
              dds::sub::LoanedSamples< T > & ls )  [inline]
```

Constructs and instance of **SharedSamples** (p. 2362) removing the ownership of the loan from the **LoanedSamples** (p. 1663).

The constructor is implicit to simplify statements like the following:

```
SharedSamples<Foo> samples = reader.take(); // reader.take() returns LoanedSamples
```

Parameters

ls	the loaned samples.

8.352.3 Member Function Documentation

8.352.3.1 begin()

```
template<typename T , template< typename Q > class DELEGATE = detail::SharedSamples>
const_iterator  dds::sub::SharedSamples< T, DELEGATE >::begin ( ) const  [inline]
```

Same as **LoanedSamples::begin()** (p. 1667)

See also

LoanedSamples::begin() (p. 1667)

8.352.3.2 end()

```
template<typename T , template< typename Q > class DELEGATE = detail::SharedSamples>
const_iterator  dds::sub::SharedSamples< T, DELEGATE >::end ( ) const  [inline]
```

Same as **LoanedSamples::end()** (p. 1668)

See also

LoanedSamples::end() (p. 1668)

8.352.3.3 operator[]()

```
template<typename T , template< typename Q > class DELEGATE = detail::SharedSamples>
DELEGATE< T >::value_type  dds::sub::SharedSamples< T, DELEGATE >::operator[] (
            size_t index ) const  [inline]
```

Same as **LoanedSamples::operator[ ](size_t)** (p. 1666)

See also

LoanedSamples::operator[ ](size_t) (p. 1666)

8.352.3.4 length()

```
template<typename T , template< typename Q > class DELEGATE = detail::SharedSamples>
uint32_t  dds::sub::SharedSamples< T, DELEGATE >::length ( ) const  [inline]
```

Returns the number of samples.

See also

LoanedSamples::length() (p. 1667)

Referenced by **dds::sub::SharedSamples< T, DELEGATE >::unpack()**.

8.352.4 Friends And Related Function Documentation

8.352.4.1 unpack() [1/3]

```
template<typename T >
void unpack (
            const  dds::sub::SharedSamples< T > & samples,
            std::vector< std::shared_ptr< const T > > & sample_vector )  [related]
```

<<***extension***>> *(p. 354)* <<***C++11***>> *(p. 353)* Unpacks a **SharedSamples** (p. 2362) collection into individual shared_ptr's in a vector

Note

`#include <rti/sub/unpack.hpp>`

This is a standalone function in the namespace **rti::sub** (p. 751)

This function creates a reference (not a copy) to each sample with valid data in a **SharedSamples** (p. 2362) container and pushes it back into a vector.

Each individual sample in the vector retains a reference to the original **SharedSamples** (p. 2362) that controls when the loan is returned. These references can be further shared. When all the references go out of scope, the loan is returned.

This can be also useful to insert samples from different calls to **read()** (p. 661)/take() into the same vector. It is however recommended to not hold these samples indefinitely, since they use internal resources.

Example:

```
dds::sub::SharedSamples<Foo> shared_samples = reader.take();
std::vector<std::shared_ptr<const Foo» sample_vector;
rti::sub::unpack(shared_samples, sample_vector);
std::cout « *sample_vector[0] « std::endl;
// ...
// Read more samples, unpack them at the end of the same vector
shared_samples = reader.take();
rti::sub::unpack(shared_samples, sample_vector);
// References to the samples can be shared freely
std::shared_ptr<const Foo> sample = sample_vector[3];
// ...
// The loans will be returned automatically
```

Note

To finalize a **DataReader** (p. 976), all the shared_ptr obtained via **unpack()** (p. 2365) need to have been released. Otherwise **DataReader::close()** (p. 1024) will fail with **dds::core::PreconditionNotMetError** (p. 1941).

Template Parameters

T	The topic-type

Parameters

samples	The collection of samples obtained from the **DataReader** (p. 976)
sample_vector	The destination where the samples are pushed back.

8.352.4.2 unpack() [2/3]

```
template<typename T >
std::vector< std::shared_ptr< const T > > unpack (
            const  dds::sub::SharedSamples< T > & samples )  [related]
```

<<***extension***>> *(p. 354)* <<***C++11***>> *(p. 353)* Unpacks a **SharedSamples** (p. 2362) collection into individual shared_ptr's in a vector

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

This overload returns a new vector instead of adding into an existing one.

See also

unpack(const dds::sub::SharedSamples<T>&, std::vector<std::shared_ptr<const T> >&) (p. 2365)

References **dds::sub::SharedSamples< T, DELEGATE >::length()**, and **rti::sub::unpack()**.

8.352.4.3 unpack() [3/3]

```
template<typename T >
std::vector< std::shared_ptr< const T > > unpack (
             dds::sub::LoanedSamples< T > && samples )  [related]
```

<<***extension***>> *(p. 354)* <<***C++11***>> *(p. 353)* Unpacks a **LoanedSamples** (p. 1663) collection into individual shared_ptr's in a vector

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

This overload is a shortcut for `unpack(SharedSamples<T>(loaned_samples))` to simplify code like the following:

```
auto sample_vector = unpack(reader.take());
```

See also

unpack(const dds::sub::SharedSamples<T>&, std::vector<std::shared_ptr<const T> >&) (p. 2365)

References **rti::sub::unpack()**.

8.353 rti::sub::SharedSamples< T > Class Template Reference

Provides access to a collection of middleware-loaned samples.

`#include <SharedSamplesImpl.hpp>`

Public Types

- typedef **SampleIterator**< T > **iterator**

  The iterator type.

8.353.1 Detailed Description

template<**typename T**>
class rti::sub::SharedSamples< **T** >

Provides access to a collection of middleware-loaned samples.

The samples in this container are loaned from the middleware and must be returned at some point.

To return the loan, use **dds::sub::LoanedSamples::return_loan** (p. 1667)

The contents of this container should not be modified and references to the samples it contains are only valid before the loan is returned.

Template Parameters

T	The data type of the contained Samples

8.353.2 Member Typedef Documentation

8.353.2.1 iterator

```
template<typename T >
typedef  SampleIterator<T>  rti::sub::SharedSamples< T >::iterator
```

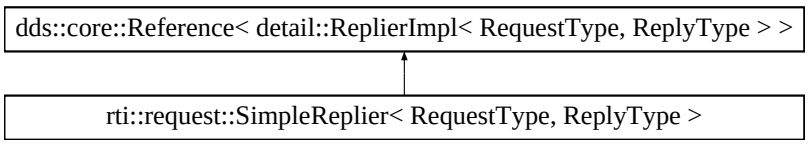
The iterator type.

8.354 rti::request::SimpleReplier< RequestType, ReplyType > Class Template Reference

<<***reference-type***>> *(p. 351)* A callback-based replier

`#include <rti/request/SimpleReplier.hpp>`

Inheritance diagram for rti::request::SimpleReplier< RequestType, ReplyType >:



Public Member Functions

- template<typename Functor >
  SimpleReplier (**dds::domain::DomainParticipant** participant, const std::string &service_name, Functor request_handler)

  *Creates a **SimpleReplier** (p. 2368).*
- template<typename Functor >
  SimpleReplier (const **ReplierParams** &params, Functor request_handler)

  *Creates a **SimpleReplier** (p. 2368) with additional parameters.*

8.354.1 Detailed Description

template<typename RequestType, typename ReplyType>
class rti::request::SimpleReplier< RequestType, ReplyType >

<<***reference-type***>> *(p. 351)* A callback-based replier

Note

A **SimpleReplier** (p. 2368) provides all the functions of a <<***reference-type***>> *(p. 351)* except **close()** (p. 1024) and retain().

A **SimpleReplier** (p. 2368) is based on a callback functor . Requests are passed to the callback, which returns a reply. The reply is directed only to the **Requester** (p. 2187) that sent the request.

SimpleRepliers are useful for simple use cases where a single reply for a request can be generated quickly, for example, looking up a table.

When more than one reply for a request can be generated or the processing is complex or needs to happen asynchronously, use a **rti::request::Replier** (p. 2168) instead.

See also

rti::request::Replier (p. 2168)

SimpleReplier example (p. 348)

In the following example, the **SimpleReplier** (p. 2368) responds to requests consisting of the built-in type **dds::core::StringTopicType** (p. 2395) with another string that prepends "Hello" to the request.

```
SimpleReplier<StringTopicType, StringTopicType> simple_replier(
    participant,
    "Test",
    [](const StringTopicType& request) {
        // If a Requester sends "World", this SimpleReplier will
        // reply with "Hello World"
        return StringTopicType("Hello " + request.data());
    }
);
```

8.354.2 Constructor & Destructor Documentation

8.354.2.1 SimpleReplier() [1/2]

```
template<typename RequestType , typename ReplyType >
template<typename Functor >
rti::request::SimpleReplier< RequestType, ReplyType >::SimpleReplier (
              dds::domain::DomainParticipant participant,
             const std::string & service_name,
             Functor request_handler )  [inline]
```

Creates a **SimpleReplier** (p. 2368).

Template Parameters

Functor	A function or function object that can receive a single parameter const RequestType& and returns a ReplyType instance by value.

Parameters

participant	The DomainParticipant that the **Replier** (p. 2168) uses to join a domain.
service_name	The service name. See **ReplierParams::service_name** (p. 2181)
request_handler	A functor that receives a request and returns a reply

8.354.2.2 SimpleReplier() [2/2]

```
template<typename RequestType , typename ReplyType >
template<typename Functor >
```

```
rti::request::SimpleReplier< RequestType, ReplyType >::SimpleReplier (
            const  ReplierParams & params,
            Functor request_handler )  [inline]
```

Creates a **SimpleReplier** (p. 2368) with additional parameters.

Template Parameters

Functor	A function or function object that can receive a single parameter const RequestType& and returns a ReplyType instance by value.

Parameters

params	The parameters used to configure the replier
request_handler	A functor that receives a request and returns a reply

8.355 rti::util::heap_monitoring::SnapshotContentFormat_def Struct Reference

Bitmap used to decide which information of the snapshot will be displayed.

`#include <rti/util/util.hpp>`

Public Types

- enum **type** {
  TOPIC ,
  FUNCTION ,
  ACTIVITY ,
  DEFAULT ,
  MINIMAL }
  The underlying `enum` type.

8.355.1 Detailed Description

Bitmap used to decide which information of the snapshot will be displayed.

The definition of the **rti::util::heap_monitoring** (p. 779) SnapshotContentFormat;

8.355.2 Member Enumeration Documentation

8.355.2.1 type

`enum` **rti::util::heap_monitoring::SnapshotContentFormat_def::type**

The underlying `enum` type.

Enumerator

TOPIC	Add the topic to the snapshot of heap monitoring.
FUNCTION	Add the function name to the snapshot of heap monitoring.
ACTIVITY	Add the activity context to the snapshot of heap monitoring. The user can select the information that will be part of the activity context by using the API **rti::config::activity_context::set_attribute_mask()** (p. 450)
DEFAULT	Add all the optional attributes to the snapshot of heap monitoring.
MINIMAL	Not add any optional attribute to the snapshot of heap monitoring.

8.356 rti::util::heap_monitoring::SnapshotOutputFormat_def Struct Reference

Specify the format of the output of the snapshot. RTI Connext.

```
#include <rti/util/util.hpp>
```

Public Types

- enum **type** {
  STANDARD ,
  COMPRESSED }

  The underlying `enum` *type.*

8.356.1 Detailed Description

Specify the format of the output of the snapshot. RTI Connext.

The definition of the **rti::util::heap_monitoring** (p. 779) SnapshotOutputFormat;

8.356.2 Member Enumeration Documentation

8.356.2.1 type

`enum` **rti::util::heap_monitoring::SnapshotOutputFormat_def::type**

The underlying `enum` type.

Enumerator

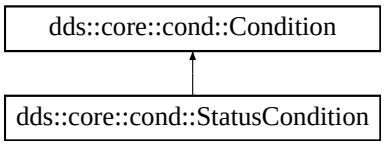
STANDARD	The output of the snapshot will be in plain text.
COMPRESSED	The output of the snapshot will be compressed using Zlib techonology. The file can be uncompressed using zlib-flate.

8.357 dds::core::cond::StatusCondition Class Reference

<<***reference-type***>> *(*p. *351)* A condition associated with each **dds::core::Entity** (p. 1520)

`#include <dds/core/cond/StatusCondition.hpp>`

Inheritance diagram for dds::core::cond::StatusCondition:



Public Member Functions

- **StatusCondition** (const **dds::core::Entity** &the_entity)

  Obtains a reference to the ***StatusCondition*** *(*p. *2372) in an entity.*
- void **enabled_statuses** (const **dds::core::status::StatusMask** &status)

  Defines the list of communication statuses that determine the trigger value.
- const **dds::core::status::StatusMask enabled_statuses** () const

  Gets the list of enabled statuses.
- **dds::core::Entity entity** () const

  Get the ***dds::core::Entity*** *(*p. *1520) associated with this* ***StatusCondition*** *(*p. *2372).*
- template<typename Functor >
  void **handler** (const Functor &func)

  <<***extension***>> *(*p. *354) Registers a custom handler with this condition.*
- void **reset_handler** ()

  <<***extension***>> *(*p. *354) Resets the handler for this condition.*

8.357.1 Detailed Description

<<***reference-type***>> *(*p. *351)* A condition associated with each **dds::core::Entity** (p. 1520)

The `trigger_value` of the **dds::core::cond::StatusCondition** (p. 2372) depends on the communication status of that entity (e.g., arrival of data, loss of information, etc.), 'filtered' by the set of `enabled_statuses` on the **dds**↩**::core::cond::StatusCondition** (p. 2372).

See also

Status Kinds (p. 429)

dds::core::cond::WaitSet (p. 2617), **dds::core::cond::Condition** (p. 948)

Listener (p. 1637)

Note

There is exactly one **StatusCondition** (p. 2372) per **Entity** (p. 1520) and one **Entity** (p. 1520) per **StatusCondition** (p. 2372).

Warning

Destroying an **Entity** (p. 1520) invalidates all references to its **StatusCondition** (p. 2372). To avoid that, always keep a reference to the **Entity** (p. 1520) while you also have one to its **StatusCondition** (p. 2372) or **retain** (p. 1526) the **Entity** (p. 1520).

8.357.2 Constructor & Destructor Documentation

8.357.2.1 StatusCondition()

```
dds::core::cond::StatusCondition::StatusCondition (
            const  dds::core::Entity & the_entity )  [inline]
```

Obtains a reference to the **StatusCondition** (p. 2372) in an entity.

Parameters

the_entity	The **Entity** (p. 1520) whose status condition we're getting a reference to. There is exactly one **StatusCondition** (p. 2372) per **Entity** (p. 1520) and one **Entity** (p. 1520) per **StatusCondition** (p. 2372).

Note

This constructor doesn't create a new **Condition** (p. 948). It obtains a reference to the **StatusCondition** (p. 2372) that each **Entity** (p. 1520) owns. You can use this constructor as many times as needed to obtain a reference to the same **StatusCondition** (p. 2372).

8.357.3 Member Function Documentation

8.357.3.1 enabled_statuses() [1/2]

```
void dds::core::cond::StatusCondition::enabled_statuses (
            const  dds::core::status::StatusMask & status )  [inline]
```

Defines the list of communication statuses that determine the trigger value.

This operation may change the `trigger_value` of the **dds::core::cond::StatusCondition** (p. 2372).

dds::core::cond::WaitSet (p. 2617) objects' behavior depends on the changes of the `trigger_value` of their attached conditions. Therefore, any **dds::core::cond::WaitSet** (p. 2617) to which the **dds::core::cond::Status↩ Condition** (p. 2372) is attached is potentially affected by this operation.

If this function is not invoked, the default list of enabled statuses includes all the statuses.

Exceptions

One	of the **Standard Exceptions** (p. 428)

8.357.3.2 enabled_statuses() [2/2]

```
const  dds::core::status::StatusMask dds::core::cond::StatusCondition::enabled_statuses ( ) const
[inline]
```

Gets the list of enabled statuses.

See also

enabled_statuses(const dds::core::status::StatusMask&) (p. 2373)

8.357.3.3 entity()

```
 dds::core::Entity dds::core::cond::StatusCondition::entity ( ) const  [inline]
```

Get the **dds::core::Entity** (p. 1520) associated with this **StatusCondition** (p. 2372).

There is exactly one **Entity** (p. 1520) associated with each **StatusCondition** (p. 2372).

8.357.3.4 handler()

```
template<typename Functor >
void handler (
             const Functor & func )
```

<<***extension***>> *(p. 354)* Registers a custom handler with this condition.

For information about condition handlers, see **this ReadCondition constructor** (p. 2134)

Note

Changing this handler will affect any other references to this **Entity** (p. 1520)'s **StatusCondition** (p. 2372).

MT Safety:

It is not safe to call **handler()** (p. 2374) or **reset_handler()** (p. 2375) concurrently or while this condition is attached to a waitset being dispatched.

8.357.3.5 reset_handler()

```
void reset_handler ( )
```

<<***extension***>> *(p. 354)* Resets the handler for this condition.

After the invocation of this method no handler will be registered with this condition.

Note

Changing this handler will affect any other references to this **Entity** (p. 1520)'s **StatusCondition** (p. 2372).

MT Safety:

It is not safe to call **handler()** (p. 2374) or **reset_handler()** (p. 2375) while this condition is attached to a waitset being dispatched.

8.358 dds::core::status::StatusMask Class Reference

A std::bitset (list) of statuses.

```
#include <State.hpp>
```

Inherits std::bitset< OMG_DDS_STATUS_COUNT >.

Public Types

- typedef std::bitset< OMG_DDS_STATUS_COUNT > **MaskType**

  The base class, a std::bitset.

Public Member Functions

- **StatusMask** ()

  *Creates **StatusMask::none()** (p. 2379)*
- **StatusMask** (uint64_t mask)

  *Creates a **StatusMask** (p. 2375) from the bits in a uint64_t value.*
- **StatusMask** (const **MaskType** &mask)

  *Creates a **StatusMask** (p. 2375) from a bitset.*

Static Public Member Functions

- static const **StatusMask** **all** ()

  All the bits are set.
- static const **StatusMask** **none** ()

  No bits are set.
- static const **StatusMask** **inconsistent_topic** ()

  Another topic exists with the same name but different characteristics.
- static const **StatusMask** **offered_deadline_missed** ()

  *The deadline that the **dds::pub::DataWriter** (p. 1138) has committed through its **dds::core::policy::Deadline** (p. 1255) was not respected for a specific instance.*
- static const **StatusMask** **requested_deadline_missed** ()

  *The deadline that the **dds::sub::DataReader** (p. 976) was expecting through its **dds::core::policy::Deadline** (p. 1255) was not respected for a specific instance.*
- static const **StatusMask** **offered_incompatible_qos** ()

  A QosPolicy value was incompatible with what was requested.
- static const **StatusMask** **requested_incompatible_qos** ()

  A QosPolicy value was incompatible with what is offered.
- static const **StatusMask** **sample_lost** ()

  A sample has been lost (i.e. was never received).
- static const **StatusMask** **sample_rejected** ()

  A (received) sample has been rejected.
- static const **StatusMask** **data_on_readers** ()

  New data is available.
- static const **StatusMask** **data_available** ()

  One or more new data samples have been received.
- static const **StatusMask** **liveliness_lost** ()

  *The liveliness that the **dds::pub::DataWriter** (p. 1138) has committed to through its **dds::core::policy::Liveliness** (p. 1645) policy was not respected, thus **dds::sub::DataReader** (p. 976) entities will consider the writer as no longer alive.*
- static const **StatusMask** **liveliness_changed** ()

  *The liveliness of one or more **dds::pub::DataWriter** (p. 1138) that were writing instances read through the **dds::sub::DataReader** (p. 976) has changed. Some **dds::pub::DataWriter** (p. 1138) have become alive or not_alive.*

- static const **StatusMask publication_matched** ()

  *The **dds::pub::DataWriter*** (p. *1138*) *has found **dds::sub::DataReader*** (p. *976*) *that matches the **dds::topic::Topic*** (p. *2474*) *and has compatible QoS.*
- static const **StatusMask subscription_matched** ()

  *The **dds::sub::DataReader*** (p. *976*) *has found **dds::pub::DataWriter*** (p. *1138*) *that matches the **dds::topic::Topic*** (p. *2474*) *and has compatible QoS.*
- static const **StatusMask reliable_writer_cache_changed** ()

  *<<**extension**>>* (p. *354*) *The number of unacknowledged samples in a reliable writer's cache has changed such that it has reached a pre-defined trigger point.*
- static const **StatusMask reliable_reader_activity_changed** ()

  *<<**extension**>>* (p. *354*) *One or more reliable readers has become active or inactive.*
- static const **StatusMask datawriter_cache** ()

  *<<**extension**>>* (p. *354*) *The status of the writer's cache.*
- static const **StatusMask datawriter_protocol** ()

  *<<**extension**>>* (p. *354*) *The status of a writer's internal protocol related metrics*
- static const **StatusMask datareader_cache** ()

  *<<**extension**>>* (p. *354*) *The status of the reader's cache.*
- static const **StatusMask datareader_protocol** ()

  *<<**extension**>>* (p. *354*) *The status of a reader's internal protocol related metrics*
- static const **StatusMask datawriter_application_acknowledgment** ()

  *<<**extension**>>* (p. *354*) *A **dds::pub::DataWriter*** (p. *1138*) *has received an application-level acknowledgment for a sample*
- static const **StatusMask datawriter_instance_replaced** ()

  *<<**extension**>>* (p. *354*) *A **dds::pub::DataWriter*** (p. *1138*) *instance has been replaced*
- static const **StatusMask service_request_accepted** ()

  *<<**extension**>>* (p. *354*) *A service request has been received for a **dds::pub::DataWriter*** (p. *1138*)
- static const **StatusMask invalid_local_advance_notice** ()

  *<<**extension**>>* (p. *354*) *The local **dds::domain::DomainParticipant*** (p. *1326*) *credential will be invalid soon.*
- static const **StatusMask sample_removed** ()

  *<<**extension**>>* (p. *354*) *A sample has been removed from a **dds::pub::DataWriter*** (p. *1138*)

Related Functions

(Note that these are not member functions.)

- template<typename STATUS >
  StatusMask get_status ()

  *Obtains the **StatusMask*** (p. *2375*) *mask associated to a status class.*

8.358.1 Detailed Description

A std::bitset (list) of statuses.

The bit-mask is an efficient and compact representation of a fixed-length list of **dds::core::status::StatusMask** (p. 2375) values.

Bits in the mask correspond to different statuses. You can choose which changes in status will trigger a callback by setting the corresponding status bits in this bit-mask and installing callbacks for each of those statuses.

The bits that are true indicate that the listener will be called back for changes in the corresponding status.

See also

Changing the listener and enabling/disabling statuses associated with it (p. 325)

8.358.2 Member Typedef Documentation

8.358.2.1 MaskType

```
typedef std::bitset<OMG_DDS_STATUS_COUNT >  dds::core::status::StatusMask::MaskType
```

The base class, a std::bitset.

8.358.3 Constructor & Destructor Documentation

8.358.3.1 StatusMask() [1/3]

```
dds::core::status::StatusMask::StatusMask ( )  [inline]
```

Creates **StatusMask::none()** (p. 2379)

Referenced by **all()**, **data_available()**, **data_on_readers()**, **datareader_cache()**, **datareader_protocol()**, **datawriter_application_acknowledgment()**, **datawriter_cache()**, **datawriter_instance_replaced()**, **datawriter_protocol()**, **inconsistent_topic()**, **invalid_local_advance_notice()**, **liveliness_changed()**, **liveliness_lost()**, **none()**, **offered_deadline_missed()**, **offered_incompatible_qos()**, **publication_matched()**, **reliable_reader_activity_changed()**, **reliable_writer_cache_changed()**, **requested_deadline_missed()**, **requested_incompatible_qos()**, **sample_lost()**, **sample_rejected()**, **sample_removed()**, **service_request_accepted()**, and **subscription_matched()**.

8.358.3.2 StatusMask() [2/3]

```
dds::core::status::StatusMask::StatusMask (
            uint64_t mask )  [inline], [explicit]
```

Creates a **StatusMask** (p. 2375) from the bits in a uint64_t value.

8.358.3.3 StatusMask() [3/3]

```
dds::core::status::StatusMask::StatusMask (
            const  MaskType & mask )  [inline]
```

Creates a **StatusMask** (p. 2375) from a bitset.

8.358.4 Member Function Documentation

8.358.4.1 all()

```
static const  StatusMask dds::core::status::StatusMask::all ( )  [inline], [static]
```

All the bits are set.

References **StatusMask()**.

8.358.4.2 none()

```
static const  StatusMask dds::core::status::StatusMask::none ( )  [inline], [static]
```

No bits are set.

References **StatusMask()**.

8.358.4.3 inconsistent_topic()

```
static const  StatusMask dds::core::status::StatusMask::inconsistent_topic ( )  [inline], [static]
```

Another topic exists with the same name but different characteristics.

Entity:

dds::topic::Topic (p. 2474)

Status:

dds::core::status::InconsistentTopicStatus (p. 1611)

Listener:

TopicListener

References **StatusMask()**.

8.358.4.4 offered_deadline_missed()

```
static const StatusMask dds::core::status::StatusMask::offered_deadline_missed ( ) [inline],
[static]
```

The deadline that the **dds::pub::DataWriter** (p. 1138) has committed through its **dds::core::policy::Deadline** (p. 1255) was not respected for a specific instance.

Entity:

dds::pub::DataWriter (p. 1138)

QoS:

DEADLINE (p. 523)

Status:

dds::core::status::OfferedDeadlineMissedStatus (p. 1871)

Listener:

dds::pub::DataWriterListener (p. 1202)

References **StatusMask()**.

8.358.4.5 requested_deadline_missed()

```
static const StatusMask dds::core::status::StatusMask::requested_deadline_missed ( ) [inline],
[static]
```

The deadline that the **dds::sub::DataReader** (p. 976) was expecting through its **dds::core::policy::Deadline** (p. 1255) was not respected for a specific instance.

Entity:

dds::sub::DataReader (p. 976)

QoS:

DEADLINE (p. 523)

Status:

dds::core::status::RequestedDeadlineMissedStatus (p. 2184)

Listener:

dds::sub::DataReaderListener (p. 1055)

References **StatusMask()**.

8.358.4.6 offered_incompatible_qos()

```
static const StatusMask dds::core::status::StatusMask::offered_incompatible_qos ( ) [inline],
[static]
```

A QosPolicy value was incompatible with what was requested.

Entity:

dds::pub::DataWriter (p. 1138)

Status:

dds::core::status::OfferedIncompatibleQosStatus (p. 1873)

Listener:

dds::pub::DataWriterListener (p. 1202)

References **StatusMask()**.

8.358.4.7 requested_incompatible_qos()

```
static const StatusMask dds::core::status::StatusMask::requested_incompatible_qos ( ) [inline],
[static]
```

A QosPolicy value was incompatible with what is offered.

Entity:

dds::sub::DataReader (p. 976)

Status:

dds::core::status::RequestedIncompatibleQosStatus (p. 2185)

Listener:

dds::sub::DataReaderListener (p. 1055)

References **StatusMask()**.

8.358.4.8 sample_lost()

```
static const StatusMask dds::core::status::StatusMask::sample_lost ( ) [inline], [static]
```

A sample has been lost (i.e. was never received).

Entity:

dds::sub::DataReader (p. 976)

Status:

dds::core::status::SampleLostStatus (p. 2303)

Listener:

dds::sub::DataReaderListener (p. 1055)

References **StatusMask()**.

8.358.4.9 sample_rejected()

```
static const StatusMask dds::core::status::StatusMask::sample_rejected ( ) [inline], [static]
```

A (received) sample has been rejected.

Entity:

dds::sub::DataReader (p. 976)

QoS:

RESOURCE_LIMITS (p. 546)

Status:

dds::core::status::SampleRejectedStatus (p. 2315)

Listener:

dds::sub::DataReaderListener (p. 1055)

References **StatusMask()**.

8.358.4.10 data_on_readers()

```
static const StatusMask dds::core::status::StatusMask::data_on_readers ( ) [inline], [static]
```

New data is available.

Entity:

dds::sub::Subscriber (p. 2410)

Listener:

dds::sub::SubscriberListener (p. 2422)

References **StatusMask()**.

8.358.4.11 data_available()

```
static const StatusMask dds::core::status::StatusMask::data_available ( ) [inline], [static]
```

One or more new data samples have been received.

Entity:

dds::sub::DataReader (p. 976)

Listener:

dds::sub::DataReaderListener (p. 1055)

References **StatusMask()**.

8.358.4.12 liveliness_lost()

```
static const StatusMask dds::core::status::StatusMask::liveliness_lost ( ) [inline], [static]
```

The liveliness that the **dds::pub::DataWriter** (p. 1138) has committed to through its **dds::core::policy::Liveliness** (p. 1645) policy was not respected, thus **dds::sub::DataReader** (p. 976) entities will consider the writer as no longer alive.

Entity:

dds::pub::DataWriter (p. 1138)

QoS:

LIVELINESS (p. 535)

Status:

dds::core::status::LivelinessLostStatus (p. 1655)

Listener:

dds::pub::DataWriterListener (p. 1202)

References **StatusMask()**.

8.358.4.13 liveliness_changed()

```
static const StatusMask dds::core::status::StatusMask::liveliness_changed ( ) [inline], [static]
```

The liveliness of one or more **dds::pub::DataWriter** (p. 1138) that were writing instances read through the **dds::sub::DataReader** (p. 976) has changed. Some **dds::pub::DataWriter** (p. 1138) have become alive or not_alive.

Entity:

dds::sub::DataReader (p. 976)

QoS:

LIVELINESS (p. 535)

Status:

dds::core::status::LivelinessChangedStatus (p. 1652)

Listener:

dds::sub::DataReaderListener (p. 1055)

References **StatusMask()**.

8.358.4.14 publication_matched()

```
static const StatusMask dds::core::status::StatusMask::publication_matched ( ) [inline], [static]
```

The **dds::pub::DataWriter** (p. 1138) has found **dds::sub::DataReader** (p. 976) that matches the **dds::topic::Topic** (p. 2474) and has compatible QoS.

Entity:

dds::pub::DataWriter (p. 1138)

Status:

dds::core::status::PublicationMatchedStatus (p. 1990)

Listener:

dds::pub::DataWriterListener (p. 1202)

References **StatusMask()**.

8.358.4.15 subscription_matched()

```
static const StatusMask dds::core::status::StatusMask::subscription_matched ( ) [inline], [static]
```

The **dds::sub::DataReader** (p. 976) has found **dds::pub::DataWriter** (p. 1138) that matches the **dds::topic::Topic** (p. 2474) and has compatible QoS.

Entity:

dds::sub::DataReader (p. 976)

Status:

dds::core::status::SubscriptionMatchedStatus (p. 2439)

Listener:

dds::sub::DataReaderListener (p. 1055)

References **StatusMask()**.

8.358.4.16 reliable_writer_cache_changed()

```
static const StatusMask dds::core::status::StatusMask::reliable_writer_cache_changed ( ) [inline],
[static]
```

<<***extension***>> *(p. 354)* The number of unacknowledged samples in a reliable writer's cache has changed such that it has reached a pre-defined trigger point.

This status is considered changed at the following times: the cache is empty (i.e., contains no unacknowledged samples), the cache is full (i.e., the sample count has reached the value specified in **dds::core::policy::ResourceLimits↩ ::max_samples** (p. 2205)), or the number of samples has reached a high (see **rti::core::policy::RtpsReliableWriter↩ Protocol::high_watermark** (p. 2233)) or low (see **rti::core::policy::RtpsReliableWriterProtocol::low_watermark** (p. 2232)) watermark.

Entity:

dds::pub::DataWriter (p. 1138)

Status:

rti::core::status::ReliableWriterCacheChangedStatus (p. 2159)

Listener:

dds::pub::DataWriterListener (p. 1202)

References **StatusMask()**.

8.358.4.17 reliable_reader_activity_changed()

```
static const StatusMask dds::core::status::StatusMask::reliable_reader_activity_changed ( ) [inline],
[static]
```

<<***extension***>> *(p. 354)* One or more reliable readers has become active or inactive.

A reliable reader is considered active by a reliable writer with which it is matched if that reader acknowledges the samples it has been sent in a timely fashion. For the definition of "timely" in this case, see **rti::core::policy::Rtps↩ ReliableWriterProtocol** (p. 2229) and **rti::core::status::ReliableReaderActivityChangedStatus** (p. 2157).

See also

rti::core::policy::RtpsReliableWriterProtocol (p. 2229)

rti::core::status::ReliableReaderActivityChangedStatus (p. 2157)

References **StatusMask()**.

8.358.4.18 datawriter_cache()

```
static const StatusMask dds::core::status::StatusMask::datawriter_cache ( ) [inline], [static]
```

<<***extension***>> *(p. 354)* The status of the writer's cache.

Changes to this status do not trigger a **dds::core::cond::StatusCondition** (p. 2372).

References **StatusMask()**.

8.358.4.19 datawriter_protocol()

```
static const StatusMask dds::core::status::StatusMask::datawriter_protocol ( ) [inline], [static]
```

<<***extension***>> *(p. 354)* The status of a writer's internal protocol related metrics

The status of a writer's internal protocol-related metrics, such as the number of samples pushed, pulled, and filtered and the status of wire protocol traffic. Changes to this status information do not trigger a **dds::core::cond::StatusCondition** (p. 2372).

References **StatusMask()**.

8.358.4.20 datareader_cache()

```
static const  StatusMask dds::core::status::StatusMask::datareader_cache ( )  [inline], [static]
```

<<***extension***>> *(p. 354)* The status of the reader's cache.

Changes to this status do not trigger a **dds::core::cond::StatusCondition** (p. 2372).

References **StatusMask()**.

8.358.4.21 datareader_protocol()

```
static const  StatusMask dds::core::status::StatusMask::datareader_protocol ( )  [inline], [static]
```

<<***extension***>> *(p. 354)* The status of a reader's internal protocol related metrics

The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic. Changes to this status do not trigger a **dds::core::cond::StatusCondition** (p. 2372).

References **StatusMask()**.

8.358.4.22 datawriter_application_acknowledgment()

```
static const  StatusMask dds::core::status::StatusMask::datawriter_application_acknowledgment ( )
[inline], [static]
```

<<***extension***>> *(p. 354)* A **dds::pub::DataWriter** (p. 1138) has received an application-level acknowledgment for a sample

Enables a **dds::pub::DataWriter** (p. 1138) callback that is called when an application-level acknowledgment from a **dds::sub::DataReader** (p. 976) is received. The callback is called for each sample that is application-level acknowledged. Changes to this status do not trigger a **dds::core::cond::StatusCondition** (p. 2372).

Entity:

dds::pub::DataWriter (p. 1138)

Listener:

dds::pub::DataWriterListener (p. 1202)

References **StatusMask()**.

8.358.4.23 datawriter_instance_replaced()

```
static const StatusMask dds::core::status::StatusMask::datawriter_instance_replaced ( ) [inline],
[static]
```

<<***extension***>> *(*p. *354)* A **dds::pub::DataWriter** (p. 1138) instance has been replaced

<<***extension***>> *(*p. *354)* A **dds::pub::DataWriter** (p. 1138) instance has been replaced

Enables a **dds::pub::DataWriter** (p. 1138) callback that is called when an instance in the writer queue is replaced.

Entity:

dds::pub::DataWriter (p. 1138)

Listener:

dds::pub::DataWriterListener (p. 1202)

References **StatusMask()**.

8.358.4.24 service_request_accepted()

```
static const StatusMask dds::core::status::StatusMask::service_request_accepted ( ) [inline],
[static]
```

<<***extension***>> *(*p. *354)* A service request has been received for a **dds::pub::DataWriter** (p. 1138)

Enables a **dds::pub::DataWriter** (p. 1138) callback that is called when a **rti::topic::ServiceRequest** (p. 2358) has been accepted and dispatched to the DataWriter.

Entity:

dds::pub::DataWriter (p. 1138)

Listener:

dds::pub::DataWriterListener (p. 1202)

References **StatusMask()**.

8.358.4.25 invalid_local_advance_notice()

```
static const StatusMask dds::core::status::StatusMask::invalid_local_advance_notice ( ) [inline],
[static]
```

<<***extension***>> *(p. 354)* The local **dds::domain::DomainParticipant** (p. 1326) credential will be invalid soon.

Enables a **dds::domain::DomainParticipant** (p. 1326) callback that is called when the local **dds::domain::Domain↩ Participant** (p. 1326) has or is about to have an invalid identity or permissions credential. Currently, this status is only triggered when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

Entity:

dds::domain::DomainParticipant (p. 1326)

Listener:

DomainParticipantListener

References **StatusMask()**.

8.358.4.26 sample_removed()

```
static const StatusMask dds::core::status::StatusMask::sample_removed ( ) [inline], [static]
```

<<***extension***>> *(p. 354)* A sample has been removed from a **dds::pub::DataWriter** (p. 1138)

Enables a **dds::pub::DataWriter** (p. 1138) callback that is called when a sample is removed from its queue.

Entity:

dds::pub::DataWriter (p. 1138)

Listener:

dds::pub::DataWriterListener (p. 1202)

References **StatusMask()**.

8.358.5 Friends And Related Function Documentation

8.358.5.1 get_status()

```
template<typename STATUS >
StatusMask get_status ( )  [related]
```

Obtains the **StatusMask** (p. 2375) mask associated to a status class.

For example:
```
using namespace dds::core::status;
StatusMask mask = get_status<PublicationMatchedStatus>();
assert(mask == StatusMask::publication_matched());
```

8.359 rti::util::StreamFlagSaver Class Reference

`#include <StreamFlagSaver.hpp>`

8.359.1 Detailed Description

RAII utility to save and restore the flags of a std::ostream

Use: void example(std::ostream& out) { **rti::util::StreamFlagSaver** (p. 2390) flag_saver (out); out << std::hex << 33; // ... } // out flags restored

8.360 rti::sub::status::StreamKind Class Reference

<<***extension***>> *(p. 354)* Indicates which stream to read from: live stream, topic-query stream or both

`#include <rti/sub/status/DataStateEx.hpp>`

Inherits std::bitset< OMG_DDS_STATE_BIT_COUNT >.

Public Types

- typedef std::bitset< OMG_DDS_STATE_BIT_COUNT > **MaskType**

  An std::bitset of StreamKinds.

Public Member Functions

- **StreamKind** (const **MaskType** &other)

  *Create an **StreamKind** (p. 2390) from MaskType.*

Static Public Member Functions

- static **StreamKind live** ()

  *Returns the **StreamKind** (p. 2390) that selects the live stream.*
- static **StreamKind topic_query** ()

  *Returns the **StreamKind** (p. 2390) that selects the topic-query stream.*
- static **StreamKind any** ()

  *Returns the **StreamKind** (p. 2390) that selects either stream.*

8.360.1 Detailed Description

<<***extension***>> *(p. 354)* Indicates which stream to read from: live stream, topic-query stream or both

See also

Topic Queries (p. 67)

8.360.2 Member Typedef Documentation

8.360.2.1 MaskType

```
typedef std::bitset<OMG_DDS_STATE_BIT_COUNT >  rti::sub::status::StreamKind::MaskType
```

An std::bitset of StreamKinds.

8.360.3 Constructor & Destructor Documentation

8.360.3.1 StreamKind()

```
rti::sub::status::StreamKind::StreamKind (
            const  MaskType & other )  [inline]
```

Create an **StreamKind** (p. 2390) from MaskType.

Parameters

other	The MaskType to create the **StreamKind** (p. 2390) with

8.360.4 Member Function Documentation

8.360.4.1 live()

```
static StreamKind rti::sub::status::StreamKind::live ( ) [inline], [static]
```

Returns the **StreamKind** (p. 2390) that selects the live stream.

8.360.4.2 topic_query()

```
static StreamKind rti::sub::status::StreamKind::topic_query ( ) [inline], [static]
```

Returns the **StreamKind** (p. 2390) that selects the topic-query stream.

8.360.4.3 any()

```
static StreamKind rti::sub::status::StreamKind::any ( ) [inline], [static]
```

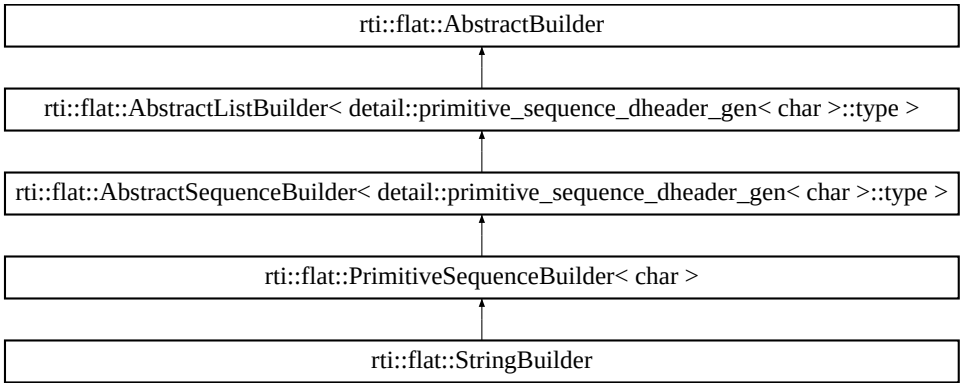
Returns the **StreamKind** (p. 2390) that selects either stream.

8.361 rti::flat::StringBuilder Class Reference

Builds a string.

```
#include <SequenceBuilders.hpp>
```

Inheritance diagram for rti::flat::StringBuilder:



Public Member Functions

- **StringBuilder** & **set_string** (const char ∗value)

  Sets the string value.
- **Offset finish** ()

  Finishes building the string.

Additional Inherited Members

8.361.1 Detailed Description

Builds a string.

A **StringBuilder** (p. 2392) only provides one method, **set_string()** (p. 2393), so it can be typically used as follows:

```
MyFlatMutableBuilder my_builder = ...;
my_builder.build_my_string().set_string("Hello!");
```

Note that by relying on the builder destructor, there is no need to call **finish()** (p. 2393) on the object returned by build↩ _my_string().

8.361.2 Member Function Documentation

8.361.2.1 set_string()

```
StringBuilder & rti::flat::StringBuilder::set_string (
            const char * value )  [inline]
```

Sets the string value.

Parameters

value	The string value to set.

Precondition

value must not be NULL.

References **rti::flat::PrimitiveSequenceBuilder< char >::add_n()**.

8.361.2.2 finish()

```
Offset rti::flat::StringBuilder::finish ( )  [inline]
```

Finishes building the string.

Returns

An Offset to the member that has been built.

See also

discard() (p. 784)

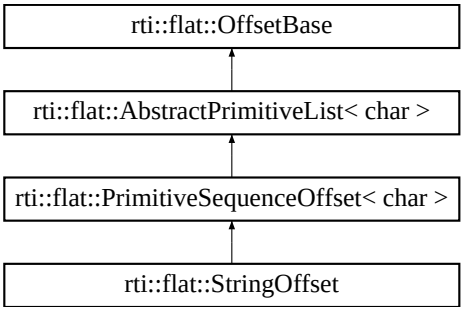
References **rti::flat::PrimitiveSequenceBuilder< char >::add_next()**.

8.362 rti::flat::StringOffset Class Reference

Offset to a string.

```
#include <SequenceOffsets.hpp>
```

Inheritance diagram for rti::flat::StringOffset:



Public Member Functions

- char * **get_string** ()
  Gets the string.
- const char * **get_string** () const
  Gets the string (const)
- unsigned int **element_count** () const
  Returns the number of characters.

8.362.1 Detailed Description

Offset to a string.

8.362.2 Member Function Documentation

8.362.2.1 get_string() [1/2]

```
char * rti::flat::StringOffset::get_string ( )  [inline]
```

Gets the string.

The string returned can be modified as long as its size doesn't change.

References **rti::flat::OffsetBase::get_buffer()**.

8.362.2.2 get_string() [2/2]

```
const char * rti::flat::StringOffset::get_string ( ) const  [inline]
```

Gets the string (const)

References **rti::flat::OffsetBase::get_buffer()**.

8.362.2.3 element_count()

```
unsigned int rti::flat::StringOffset::element_count ( ) const  [inline]
```

Returns the number of characters.

This number doesn't include the null terminating character.

References **rti::flat::PrimitiveSequenceOffset< T >::element_count()**.

8.363 dds::core::StringTopicType Class Reference

Built-in type consisting of a single character string.

```
#include <dds/core/BuiltinTopicTypes.hpp>
```

Public Member Functions

- **StringTopicType** ()

  Creates a sample containing an empty string.
- **StringTopicType** (const **dds::core::string** &the_data)

  *Creates a sample from a **dds::core::string*** (p. *435*).
- **StringTopicType** (const std::string &the_data)

  Creates a sample from a std::string.
- **StringTopicType** (const char ∗the_data)

  Creates an instance from a char∗.
- **operator std::string** () const

  Automatic conversion to std::string.
- **operator dds::core::string &** ()

  *Automatic conversion to **dds::core::string*** (p. *435*).
- **operator const dds::core::string &** () const

  *Automatic conversion to **dds::core::string*** (p. *435*).
- const **dds::core::string** & **data** () const

  Gets the string.
- **dds::core::string** & **data** ()

  Gets the string.
- void **data** (const **dds::core::string** &value)

  Sets the string.

8.363.1 Detailed Description

Built-in type consisting of a single character string.

See also

Built-in Types (p. 50)

8.363.2 Constructor & Destructor Documentation

8.363.2.1 StringTopicType() [1/4]

```
dds::core::StringTopicType::StringTopicType ( )  [inline]
```

Creates a sample containing an empty string.

8.363.2.2 StringTopicType() [2/4]

```
dds::core::StringTopicType::StringTopicType (
            const  dds::core::string & the_data )  [inline]
```

Creates a sample from a **dds::core::string** (p. 435).

Note that this constructor is implicit so you can use a **dds::core::string** (p. 435) wherever a **StringTopicType** (p. 2395) instance is expected

8.363.2.3 StringTopicType() [3/4]

```
dds::core::StringTopicType::StringTopicType (
            const std::string & the_data )  [inline]
```

Creates a sample from a std::string.

Note that this constructor is implicit so you can use an std::string wherever a **StringTopicType** (p. 2395) instance is expected, for example:

```
dds::pub::DataWriter<StringTopicType> writer(...);
std::string str = "Hello World!";
writer.write(str);
```

Parameters

the_data	The string to be copied to create this instance

8.363.2.4 StringTopicType() [4/4]

```
dds::core::StringTopicType::StringTopicType (
            const char * the_data )  [inline]
```

Creates an instance from a char∗.

Note that this constructor is implicit so you can use an char∗ wherever a **StringTopicType** (p. 2395) instance is expected, for example:

```
dds::pub::DataWriter<StringTopicType> writer(...);
writer.write("Hello, World!");
```

Parameters

the_data	The string to be copied to create this instance

8.363.3 Member Function Documentation

8.363.3.1 operator std::string()

```
dds::core::StringTopicType::operator std::string ( ) const  [inline]
```

Automatic conversion to std::string.

Returns

A copy of the string

8.363.3.2 operator dds::core::string &()

```
dds::core::StringTopicType::operator  dds::core::string & ( )  [inline]
```

Automatic conversion to **dds::core::string** (p. 435).

Returns

A reference to the string

8.363.3.3 operator const dds::core::string &()

```
dds::core::StringTopicType::operator const  dds::core::string & ( ) const  [inline]
```

Automatic conversion to **dds::core::string** (p. 435).

Returns

A const-reference to the string

8.363.3.4 data() [1/3]

```
const  dds::core::string & dds::core::StringTopicType::data ( ) const  [inline]
```

Gets the string.

Returns

A const-reference to the string

8.363.3.5 data() [2/3]

```
dds::core::string & dds::core::StringTopicType::data ( )  [inline]
```

Gets the string.

Returns

A reference to the string

8.363.3.6 data() [3/3]

```
void dds::core::StringTopicType::data (
            const  dds::core::string & value )  [inline]
```

Sets the string.

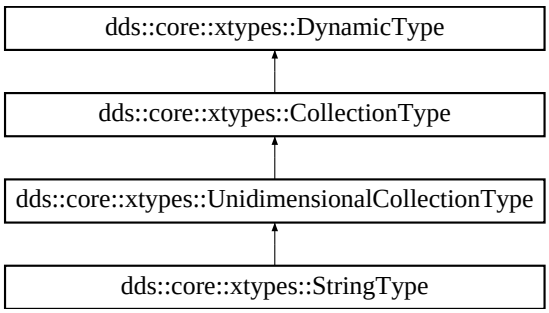
References **dds::core::Value< D >::delegate()**.

8.364 dds::core::xtypes::StringType Class Reference

<<***value-type***>> *(*p. *350)* Represents an IDL `string` type.

`#include <dds/core/xtypes/CollectionTypes.hpp>`

Inheritance diagram for dds::core::xtypes::StringType:



Public Member Functions

- **StringType** (uint32_t **bounds**)

  Creates a bounded string.

Additional Inherited Members

8.364.1 Detailed Description

<<***value-type***>> *(*p. *350)* Represents an IDL `string` type.

8.364.2 Constructor & Destructor Documentation

8.364.2.1 StringType()

```
dds::core::xtypes::StringType::StringType (
            uint32_t bounds )  [explicit]
```

Creates a bounded string.

Parameters

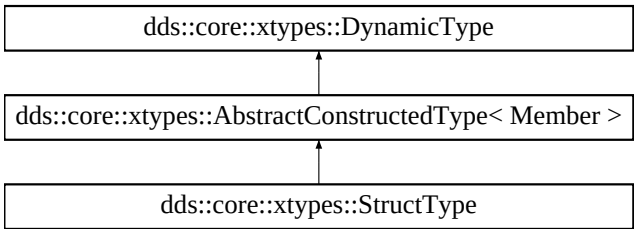
bounds	The maximum length

8.365 dds::core::xtypes::StructType Class Reference

<<***value-type***>> *(*p. *350)* Represents an IDL `struct` type

`#include <dds/core/xtypes/StructType.hpp>`

Inheritance diagram for dds::core::xtypes::StructType:



Public Member Functions

- **StructType** (const std::string & **name**)

  Creates an empty struct type.
- **StructType** (const std::string & **name**, const **StructType** & **parent**)

Creates an empty struct type with a base type.

- template<typename Container >
  StructType (const std::string &the_name, const Container &the_members)

  Creates a struct with the members in a container.
- template<typename Container >
  StructType (const std::string &the_name, const **StructType** &the_parent, const Container &the_members)

  Creates a struct with a base type and the members in a container.
- template<typename MemberIter >
  StructType (const std::string &the_name, MemberIter begin, MemberIter end)

  Creates a struct with the members in an iterator range.
- template<typename MemberIter >
  StructType (const std::string &the_name, const **StructType** &the_parent, MemberIter begin, MemberIter end)

  Creates a struct with a base type and the members in an iterator range.
- **StructType** (const std::string &the_name, **StructType** &&the_parent)

  <<***C++11***>> (p. *353) Creates an empty struct type with a base type.*
- template<typename MemberIter >
  StructType (const std::string &the_name, **StructType** &&the_parent, MemberIter begin, MemberIter end)

  <<***C++11***>> (p. *353) Creates an empty struct type with a base type and the members in an iterator range.*
- template<typename Container >
  StructType (const std::string &the_name, **StructType** &&the_parent, const Container &the_members)

  <<***C++11***>> (p. *353) Creates an empty struct type with a base type and the members in a container.*
- **StructType** (const std::string &the_name, std::initializer_list< MemberImpl > the_members)

  <<***C++11***>> (p. *353) Creates a struct with the members in an initializer_list*
- **StructType** (const std::string &the_name, const **StructType** &the_parent, std::initializer_list< MemberImpl > the_members)

  <<***C++11***>> (p. *353) Creates a struct with the members in an initializer_list and a base type*
- **StructType** (const std::string &the_name, **StructType** &&the_parent, std::initializer_list< MemberImpl > the↩_members)

  <<***C++11***>> (p. *353) Creates a struct with the members in an initializer_list and a base type*
- **StructType** & **extensibility_kind** (**dds::core::xtypes::ExtensibilityKind kind**)

  Sets the type extensibility kind.
- bool **has_parent** () const

  Indicates if this type has a base type.
- const **StructType** & **parent** () const

  Retrieves the base type.
- **MemberIndex find_member_by_id** (uint32_t id) const

  Gets the index of the member with a specific member ID.
- **StructType** & **add_member** (const MemberImpl & **member**)

  Adds a member at the end.
- template<typename Container >
  StructType & **add_members** (const Container &the_members)

  Adds all the members of a container at the end.
- **StructType** & **add_members** (std::initializer_list< MemberImpl > the_members)

  <<***C++11***>> (p. *353) Adds all the members of an initializer_list at the end*
- template<typename MemberIter >
  StructType & **add_members** (MemberIter begin, MemberIter end)

  Adds all the members in an iterator range at the end.
- **StructType** & **add_member** (MemberImpl &&the_member)

  <<***C++11***>> (p. *353) Adds a member, moving it, at the end*

Related Functions

(Note that these are not member functions.)

- template<typename... Types>
  dds::core::xtypes::StructType create_type_from_tuple (const std::string & **name**)

  *<<**C++11**>> (p. 353) <<**experimental**>> (p. 355) <<**extension**>> (p. 354) Creates a **StructType** (p. 2400) from a list of types or a std::tuple*

Additional Inherited Members

8.365.1 Detailed Description

<<***value-type***>> *(p. 350)* Represents an IDL `struct` type

Examples

Foo.hpp.

8.365.2 Constructor & Destructor Documentation

8.365.2.1 StructType() [1/12]

```
dds::core::xtypes::StructType::StructType (
            const std::string & name )
```

Creates an empty struct type.

The struct doesn't have a base type. Members can be added after creation.

Parameters

name	The name of the type

8.365.2.2 StructType() [2/12]

```
dds::core::xtypes::StructType::StructType (
            const std::string & name,
            const  StructType & parent )
```

Creates an empty struct type with a base type.

Members can be added after creation.

Parameters

name	The name of the type
parent	The base type

8.365.2.3 StructType() [3/12]

```
template<typename Container >
dds::core::xtypes::StructType::StructType (
            const std::string & the_name,
            const Container & the_members )  [inline]
```

Creates a struct with the members in a container.

The struct doesn't have a base type.

Template Parameters

Container	A container that provides the member functions **begin()** (p. 1669) and **end()** (p. 1670) to iterate over **Member** (p. 1700) elements.

Parameters

the_name	The name of the type
the_members	A container with the members for this struct type

8.365.2.4 StructType() [4/12]

```
template<typename Container >
dds::core::xtypes::StructType::StructType (
            const std::string & the_name,
            const  StructType & the_parent,
            const Container & the_members )  [inline]
```

Creates a struct with a base type and the members in a container.

Template Parameters

Container	A container that provides the member functions **begin()** (p. 1669) and **end()** (p. 1670) to iterate over **Member** (p. 1700) elements.

Parameters

the_name	The name of the type
the_parent	The base type
the_members	A container with the members for this struct type

8.365.2.5 StructType() [5/12]

```
template<typename MemberIter >
dds::core::xtypes::StructType::StructType (
            const std::string & the_name,
            MemberIter begin,
            MemberIter end )  [inline]
```

Creates a struct with the members in an iterator range.

The struct doesn't have a base type.

Template Parameters

MemberIter	A forward iterator of **Member** (p. 1700) elements

Parameters

the_name	The name of the type
begin	The beginning of the range of Members
end	The end of the range of Members

8.365.2.6 StructType() [6/12]

```
template<typename MemberIter >
dds::core::xtypes::StructType::StructType (
            const std::string & the_name,
            const  StructType & the_parent,
```

```
        MemberIter begin,
        MemberIter end )  [inline]
```

Creates a struct with a base type and the members in an iterator range.

Template Parameters

MemberIter	A forward iterator of **Member** (p. 1700) elements

Parameters

the_parent	The base type
the_name	The name of the type
begin	The beginning of the range of Members
end	The end of the range of Members

8.365.2.7 StructType() [7/12]

```
dds::core::xtypes::StructType::StructType (
             const std::string & the_name,
              StructType && the_parent )  [inline]
```

<<***C++11***>> *(p. 353)* Creates an empty struct type with a base type.

The base type is moved. This is useful to save a copy in situations like the following example:
`StructType my_type("my_type", StructType("parent_type", parent_members));`

8.365.2.8 StructType() [8/12]

```
template<typename MemberIter >
dds::core::xtypes::StructType::StructType (
             const std::string & the_name,
              StructType && the_parent,
             MemberIter begin,
             MemberIter end )  [inline]
```

<<***C++11***>> *(p. 353)* Creates an empty struct type with a base type and the members in an iterator range.

The base type is moved.

8.365.2.9 StructType() [9/12]

```
template<typename Container >
dds::core::xtypes::StructType::StructType (
             const std::string & the_name,
              StructType && the_parent,
             const Container & the_members )  [inline]
```

<<***C++11***>> *(p. 353)* Creates an empty struct type with a base type and the members in a container.

The base type is moved.

8.365.2.10 StructType() [10/12]

```
dds::core::xtypes::StructType::StructType (
            const std::string & the_name,
            std::initializer_list< MemberImpl > the_members )  [inline]
```

<<***C++11***>> *(p. 353)* Creates a struct with the members in an initializer_list

Parameters

the_name	The name of the type
the_members	An ininitalizer_list of Members

8.365.2.11 StructType() [11/12]

```
dds::core::xtypes::StructType::StructType (
            const std::string & the_name,
            const  StructType & the_parent,
            std::initializer_list< MemberImpl > the_members )  [inline]
```

<<***C++11***>> *(p. 353)* Creates a struct with the members in an initializer_list and a base type

Parameters

the_name	The name of the type
the_parent	The base type
the_members	An ininitalizer_list of Members

8.365.2.12 StructType() [12/12]

```
dds::core::xtypes::StructType::StructType (
            const std::string & the_name,
             StructType && the_parent,
            std::initializer_list< MemberImpl > the_members )  [inline]
```

<<***C++11***>> *(p. 353)* Creates a struct with the members in an initializer_list and a base type

Parameters

the_name	The name of the type
the_parent	The base type (moved)
the_members	An ininitalizer_list of Members

8.365.3 Member Function Documentation

8.365.3.1 extensibility_kind()

```
StructType & dds::core::xtypes::StructType::extensibility_kind (
            dds::core::xtypes::ExtensibilityKind kind )
```

Sets the type extensibility kind.

8.365.3.2 has_parent()

```
bool dds::core::xtypes::StructType::has_parent ( ) const
```

Indicates if this type has a base type.

8.365.3.3 parent()

```
const  StructType & dds::core::xtypes::StructType::parent ( ) const
```

Retrieves the base type.

If this type doesn't have a base type it throws **PreconditionNotMetError** (p. 1941).

See also

has_parent() (p. 2408)

8.365.3.4 find_member_by_id()

```
MemberIndex dds::core::xtypes::StructType::find_member_by_id (
            uint32_t id ) const
```

Gets the index of the member with a specific member ID.

Returns

The index (which can be passed to member(uint32_t) of the member selected by label or INVALID_INDEX if the ID doesn't exist in this type

8.365.3.5 add_member() [1/2]

```
StructType & dds::core::xtypes::StructType::add_member (
            const MemberImpl & member )
```

Adds a member at the end.

8.365.3.6 add_members() [1/3]

```
template<typename Container >
StructType & dds::core::xtypes::StructType::add_members (
            const Container & the_members )  [inline]
```

Adds all the members of a container at the end.

8.365.3.7 add_members() [2/3]

```
StructType & dds::core::xtypes::StructType::add_members (
            std::initializer_list< MemberImpl > the_members )  [inline]
```

<<***C++11***>> *(p. 353)* Adds all the members of an initializer_list at the end

8.365.3.8 add_members() [3/3]

```
template<typename MemberIter >
StructType & dds::core::xtypes::StructType::add_members (
            MemberIter begin,
            MemberIter end )  [inline]
```

Adds all the members in an iterator range at the end.

8.365.3.9 add_member() [2/2]

```
StructType & dds::core::xtypes::StructType::add_member (
            MemberImpl && the_member )  [inline]
```

<<***C++11***>> *(p. 353)* Adds a member, moving it, at the end

8.365.4 Friends And Related Function Documentation

8.365.4.1 create_type_from_tuple()

```
template<typename... Types>
dds::core::xtypes::StructType create_type_from_tuple (
            const std::string & name ) [related]
```

<<***C++11***>> *(*p. *353)* <<***experimental***>> (p. *355)* <<***extension***>> *(*p. *354)* Creates a **StructType** (p. 2400) from a list of types or a std::tuple

Note

This is a standalone function in the namespace **rti::core::xtypes** (p. 718)

Template Parameters

Types	A list of types or a std::tuple. The types must be primitive (see **dds::core::xtypes::PrimitiveType** (p. 1959)) or `std::string`.

The **StructType** (p. 2400) members are all default-created (i.e. non-key, non-optional, with default IDs...) and their names are m0, m1, m2, etc.

See also

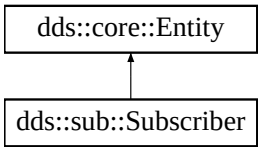
Create a DynamicType using tuples (p. 610)

8.366 dds::sub::Subscriber Class Reference

<<***reference-type***>> *(*p. *351)* A subscriber is the object responsible for actually receiving data from a subscription.

```
#include "dds/sub/Subscriber.hpp"
```

Inheritance diagram for dds::sub::Subscriber:



Public Member Functions

- **Subscriber** (const **::dds::domain::DomainParticipant** &the_participant)

  *Create a **Subscriber** (p. 2410) attached to the given DomainParticipant.*
- **Subscriber** (const **::dds::domain::DomainParticipant** &dp, const **dds::sub::qos::SubscriberQos** &the↩ _qos, **dds::sub::SubscriberListener** ∗the_listener=NULL, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Create a **Subscriber** (p. 2410) attached to the given DomainParticipant.*
- **Subscriber** (const **::dds::domain::DomainParticipant** &dp, const **dds::sub::qos::SubscriberQos** &the_qos, std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &mask= **dds::core::status↩ ::StatusMask::all**())

  *Create a **Subscriber** (p. 2410) attached to the given DomainParticipant.*
- void **notify_datareaders** ()

  *This operation invokes the operation on_data_available on the **DataReaderListener** (p. 1055) objects attached to contained **DataReader** (p. 976) entities with a DATA_AVAILABLE status that is considered changed as described in **Changes in read communication status** (p. 432).*
- void **listener** (**Listener** ∗the_listener, const **dds::core::status::StatusMask** &event_mask)

  *Attach a listener to this **Subscriber** (p. 2410).*
- **Listener** ∗ **listener** () const

  *Get the **Subscriber** (p. 2410) listener.*
- void **set_listener** (std::shared_ptr< **Listener** > the_listener, const **dds::core::status::StatusMask** &event_↩ mask)

  Sets the listener associated with this subscriber.
- void **set_listener** (std::shared_ptr< **Listener** > the_listener)

  Sets the listener associated with this subscriber.
- std::shared_ptr< **Listener** > **get_listener** () const

  *Returns the listener currently associated with this **Subscriber** (p. 2410).*
- const **dds::sub::qos::SubscriberQos** **qos** () const

  *Get the **Subscriber** (p. 2410) QoS.*
- void **qos** (const **dds::sub::qos::SubscriberQos** &the_qos)

  *Set the **Subscriber** (p. 2410) QoS.*
- **dds::sub::qos::DataReaderQos** **default_datareader_qos** () const

  *Get the default **DataReader** (p. 976) QoS.*
- **Subscriber** & **default_datareader_qos** (const **dds::sub::qos::DataReaderQos** &the_qos)

  Set the default DataReaderQoS.
- const **dds::domain::DomainParticipant** & **participant** () const

  *Return the DomainParticipant that owns this **Subscriber** (p. 2410).*
- void **close_contained_entities** ()

  *<<**extension**>> (p. 354) Closes all the entities created from this **Subscriber** (p. 2410)*

Related Functions

(Note that these are not member functions.)

- **dds::sub::Subscriber** **builtin_subscriber** (const **dds::domain::DomainParticipant** &dp)

  *Access the built-in **Subscriber** (p. 2410).*

- template<typename SubscriberForwardIterator >
  uint32_t **find_subscribers** (const **dds::domain::DomainParticipant participant**, SubscriberForwardIterator **begin**, uint32_t max_size)

  <<***extension***>> (p. *354) Retrieve all of the* ***dds::sub::Subscriber*** (p. *2410) created from this* ***dds::domain::Domain↩ Participant*** (p. *1326)*
- template<typename SubscriberBackInsertIterator >
  uint32_t **find_subscribers** (const **dds::domain::DomainParticipant participant**, SubscriberBackInsertIterator **begin**)

  <<***extension***>> (p. *354) Retrieve all of the* ***dds::sub::Subscriber*** (p. *2410) created from this* ***dds::domain::Domain↩ Participant*** (p. *1326)*
- **dds::sub::Subscriber find_subscriber** (const **dds::domain::DomainParticipant participant**, const std↩ ::string &subscriber_name)

  <<***extension***>> (p. *354) Finds a* ***Subscriber*** (p. *2410) by name*
- **dds::sub::Subscriber implicit_subscriber** (const **dds::domain::DomainParticipant** &dp)

  <<***extension***>> (p. *354) Retrieves the implicit* ***dds::sub::Subscriber*** (p. *2410) for the given* ***dds::domain::Domain↩ Participant*** (p. *1326).*

8.366.1 Detailed Description

<<***reference-type***>> (p. *351)* A subscriber is the object responsible for actually receiving data from a subscription.

QoS:

dds::sub::qos::SubscriberQos (p. 2423)

Status:

dds::core::status::StatusMask::data_on_readers() (p. 2382)

Listener:

dds::sub::SubscriberListener (p. 2422)

A subscriber acts on the behalf of one or several **dds::sub::DataReader** (p. 976) objects that are related to it. When it receives data (from the other parts of the system), it builds the list of concerned **dds::sub::DataReader** (p. 976) objects and then indicates to the application that data is available through its listener or by enabling related conditions.

The application can access the list of concerned **dds::sub::DataReader** (p. 976) objects through the operation **dds↩ ::sub::find** (p. 672) and then access the data available through operations on the **dds::sub::DataReader** (p. 976).

The following operations may be called even if the **dds::sub::Subscriber** (p. 2410) is not enabled. Other operations will the value **dds::core::NotEnabledError** (p. 1869) if called on a disabled **dds::sub::Subscriber** (p. 2410):

- **dds::core::Entity::enable** (p. 1524),
- **dds::sub::Subscriber::qos(const dds::sub::qos::SubscriberQos&)** (p. 2417), **dds::sub::Subscriber::qos() const** (p. 2417)
- **dds::sub::Subscriber::set_listener** (p. 2415), **dds::sub::Subscriber::get_listener** (p. 2416),

- **dds::core::cond::StatusCondition::StatusCondition(const dds::core::Entity&)** (p. 2373), **dds::core::Entity::status_changes()** (p. 1525)
- dds::sub::DataReader::DataReader(), **dds::sub::DataReader::close()** (p. 1024),
- **dds::sub::Subscriber::default_datareader_qos(const dds::sub::qos::DataReaderQos&)** (p. 2417), **dds::sub::Subscriber::default_datareader_qos** (p. 2417),

All operations except for **dds::sub::Subscriber::qos(const dds::sub::qos::SubscriberQos&)** (p. 2417), **dds::sub::Subscriber::qos() const** (p. 2417), **dds::sub::Subscriber::set_listener** (p. 2415), **dds::sub::Subscriber::get_listener** (p. 2416), **dds::core::Entity::enable** (p. 1524) and dds::sub::DataReader::DataReader() may fail with **dds::core::NotEnabledError** (p. 1869).

See also

Operations Allowed in Listener Callbacks (p. 1639)

dds::sub::find() (p. 672) and related functions

Entity Use Cases (p. 325)

Examples

Foo_subscriber.cxx.

8.366.2 Constructor & Destructor Documentation

8.366.2.1 Subscriber() [1/3]

```
dds::sub::Subscriber::Subscriber (
            const  ::dds::domain::DomainParticipant & the_participant )  [inline], [explicit]
```

Create a **Subscriber** (p. 2410) attached to the given DomainParticipant.

The subscriber QoS will be set to participant.default_subscriber_qos()

Parameters

the_participant	the DomainParticipant that will own this subscriber.

8.366.2.2 Subscriber() [2/3]

```
dds::sub::Subscriber::Subscriber (
            const  ::dds::domain::DomainParticipant & dp,
```

```
            const  dds::sub::qos::SubscriberQos & the_qos,
             dds::sub::SubscriberListener * the_listener = NULL,
            const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a **Subscriber** (p. 2410) attached to the given DomainParticipant.

[DEPRECATED] When using a listener, prefer the constructor that receives a shared_ptr<**Listener** (p. 1637)> instead of a regular Listener* pointer.

Parameters

dp	The DomainParticipant that will own this subscriber.
the_qos	The **qos::SubscriberQos** (p. 2423)
the_listener	The subscriber listener.
mask	Indicates which status updates the listener will receive

8.366.2.3 Subscriber() [3/3]

```
dds::sub::Subscriber::Subscriber (
            const  ::dds::domain::DomainParticipant & dp,
            const  dds::sub::qos::SubscriberQos & the_qos,
            std::shared_ptr<  Listener > the_listener,
            const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a **Subscriber** (p. 2410) attached to the given DomainParticipant.

Parameters

dp	The DomainParticipant that will own this subscriber.
the_qos	The **qos::SubscriberQos** (p. 2423)
the_listener	A shared_ptr to the listener. See **set_listener()** (p. 2415) for more information.
mask	Indicates which status updates the listener will receive

8.366.3 Member Function Documentation

8.366.3.1 notify_datareaders()

```
void dds::sub::Subscriber::notify_datareaders ( )  [inline]
```

This operation invokes the operation on_data_available on the **DataReaderListener** (p. 1055) objects attached to contained **DataReader** (p. 976) entities with a DATA_AVAILABLE status that is considered changed as described in **Changes in read communication status** (p. 432).

This operation is typically invoked from the dds::sub::SubscriberListener::on_data_on_readers operation in the **dds::sub::SubscriberListener** (p. 2422). That way the **dds::sub::SubscriberListener** (p. 2422) can delegate to the **dds::sub::DataReaderListener** (p. 1055) objects the handling of the data.

The operation will notify the data readers that have a `sample_state` of **dds::sub::status::SampleState::not_read()** (p. 2318), `view_state` of any() and `instance_state` of any().

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::NotEnabledError** (p. 1869).

8.366.3.2 listener() [1/2]

```
void dds::sub::Subscriber::listener (
              Listener * the_listener,
             const  dds::core::status::StatusMask & event_mask )  [inline]
```

Attach a listener to this **Subscriber** (p. 2410).

[DEPRECATED] The use of **`set_listener()`** (p. 2415) is recommended. Unlike this function, `set_listener` receives a `shared_ptr` which simplifies the management of listener's lifecycle.

Parameters

the_listener	The listener
event_mask	The event mask for the listener.

8.366.3.3 listener() [2/2]

```
Listener * dds::sub::Subscriber::listener ( ) const  [inline]
```

Get the **Subscriber** (p. 2410) listener.

[DEPRECATED] Use `get_listener` instead of this function.

8.366.3.4 set_listener() [1/2]

```
void dds::sub::Subscriber::set_listener (
            std::shared_ptr< Listener > the_listener,
            const dds::core::status::StatusMask & event_mask ) [inline]
```

Sets the listener associated with this subscriber.

The **Subscriber** (p. 2410) will hold a `shared_ptr` to the listener argument, ensuring that it is not deleted at least until this **Subscriber** (p. 2410) is deleted or the listener is reset with `set_listener(nullptr)`.

Warning

It's recommended that the listener implementation doesn't hold a permanent reference to the **Subscriber** (p. 2410). If it does, the application needs to manually reset the listener or manually close this **Subscriber** (p. 2410) to ensure that there is no cycle that prevents the destruction of these two objects.

Parameters

the_listener	A shared pointer to the listener to receive status updates or `nullptr` to reset the current listener and stop receiving status updates.
event_mask	A mask that indicates which status updates will be notified to the listener

8.366.3.5 set_listener() [2/2]

```
void dds::sub::Subscriber::set_listener (
            std::shared_ptr< Listener > the_listener ) [inline]
```

Sets the listener associated with this subscriber.

If `the_listener` is not `nullptr`, this overload is equivalent to:
```
subscriber.set_listener(the_listener, dds::core::status::StatusMask::all());
```

If `the_listener` is `nullptr`, it is equivalent to:
```
subscriber.set_listener(nullptr, dds::core::status::StatusMask::none());
```

Parameters

the_listener	A shared pointer to the listener to receive status updates or `nullptr` to reset the current listener and stop receiving status updates.

8.366.3.6 get_listener()

```
std::shared_ptr< Listener > dds::sub::Subscriber::get_listener ( ) const [inline]
```

Returns the listener currently associated with this **Subscriber** (p. 2410).

Returns

The shared pointer to the current listener or `nullptr` if there is currently no listener associated to this entity.

8.366.3.7 qos() [1/2]

```
const  dds::sub::qos::SubscriberQos dds::sub::Subscriber::qos ( ) const  [inline]
```

Get the **Subscriber** (p. 2410) QoS.

8.366.3.8 qos() [2/2]

```
void dds::sub::Subscriber::qos (
            const  dds::sub::qos::SubscriberQos & the_qos )  [inline]
```

Set the **Subscriber** (p. 2410) QoS.

Parameters

the_qos	the new QoS.

8.366.3.9 default_datareader_qos() [1/2]

```
dds::sub::qos::DataReaderQos dds::sub::Subscriber::default_datareader_qos ( ) const  [inline]
```

Get the default **DataReader** (p. 976) QoS.

Returns

dds::sub::qos::DataReaderQos (p. 1072)

8.366.3.10 default_datareader_qos() [2/2]

```
Subscriber & dds::sub::Subscriber::default_datareader_qos (
            const  dds::sub::qos::DataReaderQos & the_qos )  [inline]
```

Set the default DataReaderQoS.

Parameters

the_qos	The default DataReaderQoS.

8.366.3.11 participant()

```
const dds::domain::DomainParticipant & dds::sub::Subscriber::participant ( ) const [inline]
```

Return the DomainParticipant that owns this **Subscriber** (p. 2410).

Returns

const **dds::domain::DomainParticipant** (p. 1326)& The DomainParticipant that owns this subscriber

8.366.3.12 close_contained_entities()

```
void close_contained_entities ( )
```

<<***extension***>> *(p. 354)* Closes all the entities created from this **Subscriber** (p. 2410)

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Calling this function explicitly is not necessary to close a **Subscriber** (p. 2410).

Deletes all contained **dds::sub::DataReader** (p. 976) objects. This pattern is applied recursively. In this manner, the operation **dds::sub::Subscriber::close_contained_entities** (p. 2418) on the **dds::sub::Subscriber** (p. 2410) will end up deleting all the entities recursively contained in the **dds::sub::Subscriber** (p. 2410), including the **dds::sub::cond↩ ::QueryCondition** (p. 2059), **dds::sub::cond::ReadCondition** (p. 2133), and **rti::sub::TopicQuery** (p. 2516) objects belonging to the contained **dds::sub::DataReader** (p. 976).

The operation will fail with **dds::core::PreconditionNotMetError** (p. 1941) if any of the contained entities is in a state where it cannot be deleted. This will occur, for example, if a contained **dds::sub::DataReader** (p. 976) cannot be deleted because the application has called a **dds::sub::DataReader::read** (p. 996) or **dds::sub::DataReader::take** (p. 997) operation and the loan has not been returned.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::PreconditionNotMetError** (p. 1941)

8.366.4 Friends And Related Function Documentation

8.366.4.1 builtin_subscriber()

```
dds::sub::Subscriber builtin_subscriber (
            const  dds::domain::DomainParticipant & dp )  [related]
```

Access the built-in **Subscriber** (p. 2410).

Each **dds::domain::DomainParticipant** (p. 1326) contains several built-in **dds::topic::Topic** (p. 2474) objects as well as corresponding **dds::sub::DataReader** (p. 976) objects to access them. All of these **dds::sub::DataReader** (p. 976) objects belong to a single built-in **dds::sub::Subscriber** (p. 2410).

The built-in Topics are used to communicate information about other **dds::domain::DomainParticipant** (p. 1326), **dds::topic::Topic** (p. 2474), **dds::sub::DataReader** (p. 976), and **dds::pub::DataWriter** (p. 1138) objects.

The built-in subscriber is created when this operation is called for the first time. The built-in subscriber is deleted automatically when the **dds::domain::DomainParticipant** (p. 1326) is deleted.

Returns

The built-in **dds::sub::Subscriber** (p. 2410) singleton.

See also

dds::topic::SubscriptionBuiltinTopicData (p. 2428)

dds::topic::PublicationBuiltinTopicData (p. 1977)

dds::topic::ParticipantBuiltinTopicData (p. 1908)

dds::topic::TopicBuiltinTopicData (p. 2493)

Parameters

dp	The DomainParticipant that the built-in subscriber belongs to.

8.366.4.2 find_subscribers() [1/2]

```
template<typename SubscriberForwardIterator >
uint32_t find_subscribers (
            const  dds::domain::DomainParticipant participant,
            SubscriberForwardIterator begin,
            uint32_t max_size )  [related]
```

<<***extension***>> *(p. 354)* Retrieve all of the **dds::sub::Subscriber** (p. 2410) created from this **dds::domain::DomainParticipant** (p. 1326)

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

SubscriberForwardIterator	Type of the forward iterator passed into this function, whose value_type must be **dds::sub::Subscriber** (p. 2410).

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Publishers belong to
begin	A forward iterator to the position in the destination container where to begin copying the found Subscribers into
max_size	The maximum size of Subscribers to add

Returns

The number of found Subscribers

References **rti::sub::begin()**.

8.366.4.3 find_subscribers() [2/2]

```
template<typename SubscriberBackInsertIterator >
uint32_t find_subscribers (
            const  dds::domain::DomainParticipant participant,
            SubscriberBackInsertIterator begin )  [related]
```

<<***extension***>> *(p. 354)* Retrieve all of the **dds::sub::Subscriber** (p. 2410) created from this **dds::domain::DomainParticipant** (p. 1326)

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

AnyDataReaderBackInsertIterator	Type of the back-inserting iterator passed into this function, whose value_type must be **dds::sub::Subscriber** (p. 2410).

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Subscribers belong to
begin	A back-inserting iterator to the position in the destination container to insert the found Subscribers into

Returns

The number of found Subscribers

References **rti::sub::begin()**.

8.366.4.4 find_subscriber()

```
dds::sub::Subscriber find_subscriber (
            const  dds::domain::DomainParticipant participant,
            const std::string & subscriber_name )  [related]
```

<<***extension***>> *(p. 354)* Finds a **Subscriber** (p. 2410) by name

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

Template Parameters

AnyDataReaderBackInsertIterator	Type of the back-inserting iterator passed into this function, whose value_type must be **dds::sub::AnyDataReader** (p. 807).

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Subscribers belong to
subscriber_name	The name of the **Subscriber** (p. 2410)

Returns

A valid reference if the a subscriber with that name exists or a reference equals to **dds::core::null** (p. 438) otherwise.

8.366.4.5 implicit_subscriber()

```
dds::sub::Subscriber implicit_subscriber (
            const  dds::domain::DomainParticipant & dp )  [related]
```

<<***extension***>> *(p. 354)* Retrieves the implicit **dds::sub::Subscriber** (p. 2410) for the given **dds::domain::DomainParticipant** (p. 1326).

Note

This is a standalone function in the namespace **rti::sub** (p. 751)

If an implicit **Subscriber** (p. 2410) does not already exist, this creates one.

The implicit **Subscriber** (p. 2410) is created with default **dds::sub::qos::SubscriberQos** (p. 2423) and no listener. When a DomainParticipant is deleted, if there are no attached **dds::sub::DataReader** (p. 976) that belong to the implicit **Subscriber** (p. 2410), the implicit **Subscriber** (p. 2410) will be implicitly deleted.

MT Safety:

UNSAFE. it is not safe to create the implicit subscriber while another thread may be simultaneously calling **dds::domain::DomainParticipant::default_subscriber_qos(const dds::sub::qos::SubscriberQos & qos)** (p. 1343).

Parameters

dp	The DomainParticipant that the implicit subscriber belongs to.

Returns

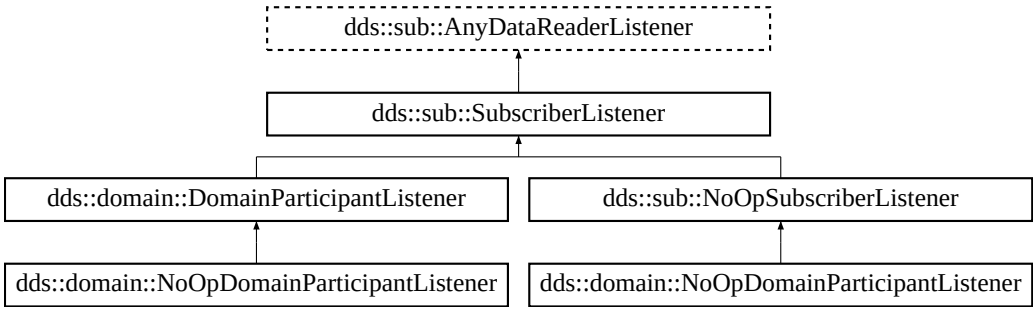
The implicit **Subscriber** (p. 2410)

8.367 dds::sub::SubscriberListener Class Reference

The listener to notify status changes for a **dds::sub::Subscriber** (p. 2410).

`#include <dds/sub/SubscriberListener.hpp>`

Inheritance diagram for dds::sub::SubscriberListener:



Additional Inherited Members

8.367.1 Detailed Description

The listener to notify status changes for a **dds::sub::Subscriber** (p. 2410).

A **SubscriberListener** (p. 2422) contains exactly the same methods as a **AnyDataReaderListener** (p. 812), since it can receive status changes from any of its contained DataReaders.

Entity:

dds::sub::Subscriber (p. 2410)

Status:

dds::core::status::StatusMask::data_available() (p. 2383);
dds::core::status::StatusMask::data_on_readers() (p. 2382);
dds::core::status::StatusMask::liveliness_changed() (p. 2383), **dds::core::status::LivelinessChanged↩ Status** (p. 1652);
dds::core::status::StatusMask::requested_deadline_missed() (p. 2380), **dds::core::status::Requested↩ DeadlineMissedStatus** (p. 2184);
dds::core::status::StatusMask::requested_incompatible_qos() (p. 2381), **dds::core::status::Requested↩ IncompatibleQosStatus** (p. 2185);
dds::core::status::StatusMask::sample_lost() (p. 2381), **dds::core::status::SampleLostStatus** (p. 2303);
dds::core::status::StatusMask::sample_rejected() (p. 2382), **dds::core::status::SampleRejectedStatus** (p. 2315);
dds::core::status::StatusMask::subscription_matched() (p. 2384), **dds::core::status::Subscription↩ MatchedStatus** (p. 2439);

See also

Listener (p. 1637)

Status Kinds (p. 429)

Operations Allowed in Listener Callbacks (p. 1639)

See also

NoOpSubscriberListener (p. 1864)

8.368 dds::sub::qos::SubscriberQos Class Reference

<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::sub::Subscriber** (p. 2410) supports

```
#include "dds/sub/qos/SubscriberQos.hpp"
```

Public Member Functions

- template<typename POLICY >
  const POLICY & **policy** () const

  Gets a QoS policy by const reference.
- template<typename POLICY >
  POLICY & **policy** ()

  Gets a QoS policy by reference.

Related Functions

(Note that these are not member functions.)

- std::string **to_string** (const **SubscriberQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::sub::qos::SubscriberQos*** *(p. 2423)*
- std::string **to_string** (const **SubscriberQos** &qos, const **SubscriberQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::sub::qos::SubscriberQos*** *(p. 2423)*
- std::string **to_string** (const **SubscriberQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::sub::qos::SubscriberQos*** *(p. 2423)*
- std::ostream & **operator**<< (std::ostream &out, const rti::sub::qos::SubscriberQos &qos)

  <<***extension***>> *(p. 354) Prints a* ***dds::sub::qos::SubscriberQos*** *(p. 2423) to an output stream.*

8.368.1 Detailed Description

<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::sub::Subscriber** (p. 2410) supports

You must set certain members in a consistent manner:

length of **dds::core::policy::GroupData::value** (p. 1593) <= **rti::core::policy::DomainParticipantResourceLimits::subscriber_group_data_max_length** (p. 1415)

length of **dds::core::policy::Partition::name** (p. 1926) <= **rti::core::policy::DomainParticipantResourceLimits::max_partitions** (p. 1417)

combined number of characters (including terminating 0) in **dds::core::policy::Partition::name** (p. 1926) <= **rti::core::policy::DomainParticipantResourceLimits::max_partition_cumulative_characters** (p. 1417)

If any of the above are not true, **dds::sub::Subscriber::qos(const dds::sub::qos::SubscriberQos&)** (p. 2417) will fail with **dds::core::InconsistentPolicyError** (p. 1610)

See also

Qos Use Cases (p. 602)

8.368.2 Member Function Documentation

8.368.2.1 policy() [1/2]

```
template<typename POLICY >
const POLICY & dds::sub::qos::SubscriberQos::policy ( ) const
```

Gets a QoS policy by const reference.

See also

policy() (p. 2425)

8.368.2.2 policy() [2/2]

```
template<typename POLICY >
POLICY & dds::sub::qos::SubscriberQos::policy ( )
```

Gets a QoS policy by reference.

Template Parameters

Policy	One of the **Subscriber** (p. 2410) QoS policies: • **dds::core::policy::Presentation** (p. 1942), • **dds::core::policy::Partition** (p. 1923), • **dds::core::policy::GroupData** (p. 1591), • **dds::core::policy::EntityFactory** (p. 1527), • **rti::core::policy::EntityName** (p. 1530)

See also

DDSQosModule_set_qos

8.368.3 Friends And Related Function Documentation

8.368.3.1 to_string() [1/3]

```
std::string to_string (
             const  SubscriberQos & qos,
             const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::sub::qos::SubscriberQos** (p. 2423)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
SubscriberQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for SubscriberQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
SubscriberQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

8.368.3.2 to_string() [2/3]

```
std::string to_string (
             const  SubscriberQos & qos,
             const  SubscriberQos & base,
             const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::sub::qos::SubscriberQos** (p. 2423)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to
	print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019)
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

8.368.3.3 to_string() [3/3]

```
std::string to_string (
            const  SubscriberQos & qos,
            const  rti::core::qos_print_all_t & qos_print_all,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(*p. *354**)* Obtains a string representation of the **dds::sub::qos::SubscriberQos** (p. 2423)

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
```
to_string(qos, rti::core::qos_print_all);
```

Returns

The string representation of the qos

8.368.3.4 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const rti::sub::qos::SubscriberQos & qos )  [related]
```

<<***extension***>> *(*p. *354**)* Prints a **dds::sub::qos::SubscriberQos** (p. 2423) to an output stream.

8.369 dds::topic::SubscriptionBuiltinTopicData Class Reference

Entry created when a **dds::sub::DataReader** (p. 976) is discovered in association with its **dds::sub::Subscriber** (p. 2410).

```
#include <dds/topic/BuiltinTopic.hpp>
```

Public Member Functions

- **TSubscriptionBuiltinTopicData** ()

  *Create a default **SubscriptionBuiltinTopicData** (p. 2428).*
- const **dds::topic::BuiltinTopicKey** & **key** () const

  Get the DCPS key to distinguish entries.
- const **dds::topic::BuiltinTopicKey** & **participant_key** () const

  *Get the DCPS key of the **dds::domain::DomainParticipant** (p. 1326) to which the **dds::sub::DataReader** (p. 976) belongs.*
- const **dds::core::string** & **topic_name** () const

  *Get the name of the related **dds::topic::Topic** (p. 2474).*
- const **dds::core::string** & **type_name** () const

  *Get the name of the type attached to the **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::Durability** & **durability** () const

  *Get the **dds::core::policy::Durability** (p. 1433) policy of the corresponding **dds::sub::DataReader** (p. 976).*
- const **dds::core::policy::Deadline** & **deadline** () const

  *Get the **dds::core::policy::Deadline** (p. 1255) policy of the corresponding **dds::sub::DataReader** (p. 976).*
- const **dds::core::policy::LatencyBudget** & **latency_budget** () const

  *Get the **dds::core::policy::LatencyBudget** (p. 1631) policy of the corresponding **dds::sub::DataReader** (p. 976).*
- const **dds::core::policy::Liveliness** & **liveliness** () const

  *Get the **dds::core::policy::Liveliness** (p. 1645) policy of the corresponding **dds::sub::DataReader** (p. 976).*
- const **dds::core::policy::Reliability** & **reliability** () const

  *Get the **dds::core::policy::Reliability** (p. 2149) policy of the corresponding **dds::sub::DataReader** (p. 976).*
- const **dds::core::policy::Ownership** & **ownership** () const

  *Get the **dds::core::policy::Ownership** (p. 1899) policy of the corresponding **dds::sub::DataReader** (p. 976).*
- const **dds::core::policy::DestinationOrder** & **destination_order** () const

  *Get the **dds::core::policy::DestinationOrder** (p. 1258) policy of the corresponding **dds::sub::DataReader** (p. 976).*
- const **dds::core::policy::UserData** & **user_data** () const

  *Get the **dds::core::policy::UserData** (p. 2591) policy of the corresponding **dds::sub::DataReader** (p. 976).*
- const **dds::core::policy::TimeBasedFilter** & **time_based_filter** () const

  *Get the **dds::core::policy::TimeBasedFilter** (p. 2469) policy of the corresponding **dds::sub::DataReader** (p. 976).*
- const **dds::core::policy::Presentation** & **presentation** () const

  *Get the **dds::core::policy::Presentation** (p. 1942) policy of the corresponding **dds::sub::DataReader** (p. 976).*
- const **dds::core::policy::Partition** & **partition** () const

  *Get the **dds::core::policy::Partition** (p. 1923) policy of the corresponding **dds::sub::DataReader** (p. 976).*
- const **dds::core::policy::TopicData** & **topic_data** () const

  *Get the **dds::core::policy::TopicData** (p. 2500) policy of the related **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::GroupData** & **group_data** () const

  *Get the **dds::core::policy::GroupData** (p. 1591) policy of the **dds::sub::Subscriber** (p. 2410) to which the **dds::sub::DataReader** (p. 976) belongs.*

- const **dds::core::policy::DataRepresentation** & **representation** () const

  *Get the **dds::core::policy::DataRepresentation*** (p. *1113*) *policy of the corresponding **dds::sub::DataReader*** (p. *976*).
- const **dds::core::policy::DataTag** & **data_tag** () const

  *Get the **dds::core::policy::DataTag*** (p. *1132*) *policy of the corresponding **dds::sub::DataReader*** (p. *976*).
- **dds::core::optional**< **dds::core::xtypes::DynamicType** > **type** () const

  <<***extension***>> (p. *354*) *Get the type*
- const **dds::core::optional**< **dds::core::xtypes::DynamicType** > & **get_type_no_copy** () const

  <<***extension***>> (p. *354*) *Get the type by reference if possible*
- const **dds::topic::BuiltinTopicKey** & **subscriber_key** () const

  <<***extension***>> (p. *354*) *Get the DCPS key of the subscriber to which the DataReader belongs.*
- const **rti::core::policy::Property** & **property** () const

  <<***extension***>> (p. *354*) *Get the Property policy of the corresponding DataReader*
- const **dds::core::vector**< **rti::core::Locator** > & **unicast_locators** () const

  <<***extension***>> (p. *354*) *Get the custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.*
- const **dds::core::vector**< **rti::core::Locator** > & **multicast_locators** () const

  <<***extension***>> (p. *354*) *Get the custom multicast locators that the endpoint can specify. The default locators will be used if this is not specified.*
- const **rti::core::ContentFilterProperty** & **content_filter_property** () const

  <<***extension***>> (p. *354*) *This provides all the required information to enable content filtering on the Writer side.*
- const **rti::core::Guid** & **virtual_guid** () const

  <<***extension***>> (p. *354*) *Gets the virtual GUID associated to the DataReader.*
- const **rti::core::ProtocolVersion** & **rtps_protocol_version** () const

  <<***extension***>> (p. *354*) *Get the version of the RTPS wire protocol used.*
- const **rti::core::VendorId** & **rtps_vendor_id** () const

  <<***extension***>> (p. *354*) *Get the ID of vendor implementing the RTPS wire protocol*
- const **rti::core::ProductVersion** & **product_version** () const

  <<***extension***>> (p. *354*) *Gets the version of RTI Connext*
- bool **disable_positive_acks** () const

  <<***extension***>> (p. *354*) *Get whether or not the corresponding **dds::sub::DataReader*** (p. *976*) *sends positive acknowledgments for reliability.*
- const **rti::core::policy::EntityName** & **subscription_name** () const

  <<***extension***>> (p. *354*) *Get the subscription name and role name.*
- const **rti::topic::trust::EndpointTrustProtectionInfo** & **trust_protection_info** () const

  <<***extension***>> (p. *354*) *Get the Trust Plugins information associated with the discovered DataReader.*
- const **rti::topic::trust::EndpointTrustAlgorithmInfo** & **trust_algorithm_info** () const

  <<***extension***>> (p. *354*) *Get the Trust plugins algorithms associated with the discovered DataReader.*
- const **dds::topic::BuiltinTopicKey** & **related_datawriter_key** () const

  *Get the **dds::topic::BuiltinTopicKey*** (p. *905*) *of the related **dds::pub::DataWriter*** (p. *1138*) *of this DataReader.*
- const **rti::core::policy::Service** & **service** () const

  <<***extension***>> (p. *354*) *Get the Service policy of the corresponding DataReader.*

8.369.1 Detailed Description

Entry created when a **dds::sub::DataReader** (p. 976) is discovered in association with its **dds::sub::Subscriber** (p. 2410).

Data associated with the built-in topic **dds::topic::subscription_topic_name()** (p. 444). It contains QoS policies and additional information that apply to the remote **dds::sub::DataReader** (p. 976) the related **dds::sub::Subscriber** (p. 2410).

8.369.2 Member Function Documentation

8.369.2.1 TSubscriptionBuiltinTopicData()

```
dds::topic::SubscriptionBuiltinTopicData::TSubscriptionBuiltinTopicData ( )  [inline]
```

Create a default **SubscriptionBuiltinTopicData** (p. 2428).

8.369.2.2 key()

```
const  dds::topic::BuiltinTopicKey & dds::topic::SubscriptionBuiltinTopicData::key ( ) const  [inline]
```

Get the DCPS key to distinguish entries.

Returns

The key

8.369.2.3 participant_key()

```
const  dds::topic::BuiltinTopicKey & dds::topic::SubscriptionBuiltinTopicData::participant_key ( )
const  [inline]
```

Get the DCPS key of the **dds::domain::DomainParticipant** (p. 1326) to which the **dds::sub::DataReader** (p. 976) belongs.

Returns

The DomainParticipant's key

8.369.2.4 topic_name()

```
const  dds::core::string & dds::topic::SubscriptionBuiltinTopicData::topic_name ( ) const  [inline]
```

Get the name of the related **dds::topic::Topic** (p. 2474).

Returns

The topic name

8.369.2.5 type_name()

```
const dds::core::string & dds::topic::SubscriptionBuiltinTopicData::type_name ( ) const [inline]
```

Get the name of the type attached to the **dds::topic::Topic** (p. 2474).

Returns

The type name

8.369.2.6 durability()

```
const dds::core::policy::Durability & dds::topic::SubscriptionBuiltinTopicData::durability ( )
const [inline]
```

Get the **dds::core::policy::Durability** (p. 1433) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The Durability policy

8.369.2.7 deadline()

```
const dds::core::policy::Deadline & dds::topic::SubscriptionBuiltinTopicData::deadline ( ) const
[inline]
```

Get the **dds::core::policy::Deadline** (p. 1255) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The Deadline policy

8.369.2.8 latency_budget()

```
const dds::core::policy::LatencyBudget & dds::topic::SubscriptionBuiltinTopicData::latency_budget
( ) const [inline]
```

Get the **dds::core::policy::LatencyBudget** (p. 1631) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The LatencyBudget policy

8.369.2.9 liveliness()

```
const dds::core::policy::Liveliness & dds::topic::SubscriptionBuiltinTopicData::liveliness ( )
const [inline]
```

Get the **dds::core::policy::Liveliness** (p. 1645) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The Liveliness policy

8.369.2.10 reliability()

```
const dds::core::policy::Reliability & dds::topic::SubscriptionBuiltinTopicData::reliability ( )
const [inline]
```

Get the **dds::core::policy::Reliability** (p. 2149) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The Reliability policy

8.369.2.11 ownership()

```
const dds::core::policy::Ownership & dds::topic::SubscriptionBuiltinTopicData::ownership ( )
const [inline]
```

Get the **dds::core::policy::Ownership** (p. 1899) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The Ownership policy

8.369.2.12 destination_order()

```
const dds::core::policy::DestinationOrder & dds::topic::SubscriptionBuiltinTopicData::destination↩
_order ( ) const [inline]
```

Get the **dds::core::policy::DestinationOrder** (p. 1258) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The DestinationOrder policy

8.369.2.13 user_data()

```
const dds::core::policy::UserData & dds::topic::SubscriptionBuiltinTopicData::user_data ( ) const
[inline]
```

Get the **dds::core::policy::UserData** (p. 2591) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The UserData policy

8.369.2.14 time_based_filter()

```
const dds::core::policy::TimeBasedFilter & dds::topic::SubscriptionBuiltinTopicData::time_based↩
_filter ( ) const [inline]
```

Get the **dds::core::policy::TimeBasedFilter** (p. 2469) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The TimeBasedFilter policy

8.369.2.15 presentation()

```
const dds::core::policy::Presentation & dds::topic::SubscriptionBuiltinTopicData::presentation (
) const [inline]
```

Get the **dds::core::policy::Presentation** (p. 1942) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The Presentation policy

8.369.2.16 partition()

```
const dds::core::policy::Partition & dds::topic::SubscriptionBuiltinTopicData::partition ( )
const [inline]
```

Get the **dds::core::policy::Partition** (p. 1923) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The Partition policy

8.369.2.17 topic_data()

```
const dds::core::policy::TopicData & dds::topic::SubscriptionBuiltinTopicData::topic_data ( )
const [inline]
```

Get the **dds::core::policy::TopicData** (p. 2500) policy of the related **dds::topic::Topic** (p. 2474).

Returns

The TopicData policy

8.369.2.18 group_data()

```
const dds::core::policy::GroupData & dds::topic::SubscriptionBuiltinTopicData::group_data ( )
const [inline]
```

Get the **dds::core::policy::GroupData** (p. 1591) policy of the **dds::sub::Subscriber** (p. 2410) to which the **dds::sub::DataReader** (p. 976) belongs.

Returns

The GroupData policy

8.369.2.19 representation()

```
const dds::core::policy::DataRepresentation & dds::topic::SubscriptionBuiltinTopicData::representation
( ) const [inline]
```

Get the **dds::core::policy::DataRepresentation** (p. 1113) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The DataRepresentation policy

8.369.2.20 data_tag()

```
const dds::core::policy::DataTag & dds::topic::SubscriptionBuiltinTopicData::data_tag ( ) const
[inline]
```

Get the **dds::core::policy::DataTag** (p. 1132) policy of the corresponding **dds::sub::DataReader** (p. 976).

Returns

The DataTag policy

8.369.2.21 type()

```
dds::core::optional< dds::core::xtypes::DynamicType > type ( ) const
```

<<***extension***>> *(p. 354)* Get the type

Note

This is not a lightweight getter–obtaining the type may require some processing so it is recommended that you keep the value if you need to use it multiple times rather than look it up every time. See also **get_type_no_copy()** (p. 2435).

Returns

The type or an unset optional value if the type is not available

8.369.2.22 get_type_no_copy()

```
const dds::core::optional< dds::core::xtypes::DynamicType > & get_type_no_copy ( ) const
```

<<***extension***>> *(p. 354)* Get the type by reference if possible

Exceptions

dds::core::PreconditionNotMet	If the type is available but is not in a format that is directly accessible. In that case it needs some extra processing and has to be accessed using **type()** (p. 2434). You can also ensure that it is in the right format by setting **rti::core::policy::↩ DomainParticipantResourceLimits::type_code_max_serialized_length()** (p. 1418) to 0.

Returns

The type or an unset optional value if the type is not available

See also

type() (p. 2434)

8.369.2.23 subscriber_key()

```
const dds::topic::BuiltinTopicKey & subscriber_key ( ) const
```

<<***extension***>> *(p. 354)* Get the DCPS key of the subscriber to which the DataReader belongs.

8.369.2.24 property()

```
const rti::core::policy::Property & property ( ) const
```

<<***extension***>> *(p. 354)* Get the Property policy of the corresponding DataReader

Returns

8.369.2.25 unicast_locators()

```
const dds::core::vector< rti::core::Locator > & unicast_locators ( ) const
```

<<***extension***>> *(p. 354)* Get the custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.

8.369.2.26 multicast_locators()

```
const dds::core::vector< rti::core::Locator > & multicast_locators ( ) const
```

<<***extension***>> *(p. 354)* Get the custom multicast locators that the endpoint can specify. The default locators will be used if this is not specified.

8.369.2.27 content_filter_property()

```
const rti::core::ContentFilterProperty & content_filter_property ( ) const
```

<<***extension***>> *(p. 354)* This provides all the required information to enable content filtering on the Writer side.

8.369.2.28 virtual_guid()

```
const rti::core::Guid & virtual_guid ( ) const
```

<<***extension***>> *(p. 354)* Gets the virtual GUID associated to the DataReader.

8.369.2.29 rtps_protocol_version()

```
const rti::core::ProtocolVersion & rtps_protocol_version ( ) const
```

<<***extension***>> *(p. 354)* Get the version of the RTPS wire protocol used.

8.369.2.30 rtps_vendor_id()

```
const rti::core::VendorId & rtps_vendor_id ( ) const
```

<<***extension***>> *(p. 354)* Get the ID of vendor implementing the RTPS wire protocol

8.369.2.31 product_version()

```
const rti::core::ProductVersion & product_version ( ) const
```

<<***extension***>> *(p. 354)* Gets the version of RTI Connext

Returns

8.369.2.32 disable_positive_acks()

```
bool disable_positive_acks ( ) const
```

<<***extension***>> *(p. 354)* Get whether or not the corresponding **dds::sub::DataReader** (p. 976) sends positive acknowledgments for reliability.

Returns

false if positive acknowledgments are disabled, true otherwise

8.369.2.33 subscription_name()

```
const rti::core::policy::EntityName & subscription_name ( ) const
```

<<***extension***>> *(p. 354)* Get the subscription name and role name.

8.369.2.34 trust_protection_info()

```
const  rti::topic::trust::EndpointTrustProtectionInfo & trust_protection_info ( ) const
```

<<***extension***>> *(p. 354)* Get the Trust Plugins information associated with the discovered DataReader.

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged data and metadata.

trust_protection_info contains information about how RTPS wire serialization, discovery, and liveliness interact with the loaded Trust Plugins. Two endpoints will not match if their trust_protection_info is incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.369.2.35 trust_algorithm_info()

```
const  rti::topic::trust::EndpointTrustAlgorithmInfo & trust_algorithm_info ( ) const
```

<<***extension***>> *(p. 354)* Get the Trust plugins algorithms associated with the discovered DataReader.

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged data and metadata.

trust_algorithm_info contains information about what algorithms the loaded Trust Plugins are running. Two endpoints will not match if their trust_algorithm_info are incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.369.2.36 related_datawriter_key()

```
const  dds::topic::BuiltinTopicKey & related_datawriter_key ( ) const
```

Get the **dds::topic::BuiltinTopicKey** (p. 905) of the related **dds::pub::DataWriter** (p. 1138) of this DataReader.

This field will be set to something other than **dds::topic::BuiltinTopicKey::unknown()** (p. 906) only when the discovered DataReader is a part of Remote Procedure Call (RPC) or Request-Reply endpoints.

8.369.2.37 service()

```
const  rti::core::policy::Service & service ( ) const
```

<<***extension***>> *(p. 354)* Get the Service policy of the corresponding DataReader.

8.370 dds::core::status::SubscriptionMatchedStatus Class Reference

Information about the status **dds::core::status::StatusMask::subscription_matched()** (p. 2384)

`#include <TStatus.hpp>`

Public Member Functions

- int32_t **total_count** () const

  *The total cumulative number of times that this **dds::sub::DataReader** (p. 976) discovered a "match" with a **dds::pub::DataWriter** (p. 1138).*
- int32_t **total_count_change** () const

  The changes in total_count since the last time the listener was called or the status was read.
- int32_t **current_count** () const

  *The current number of DataWriters with which the **dds::sub::DataReader** (p. 976) is matched.*
- int32_t **current_count_change** () const

  The change in current_count since the last time the listener was called or the status was read.
- const **dds::core::InstanceHandle last_publication_handle** () const

  *This **InstanceHandle** (p. 1612) can be used to look up which remote **dds::pub::DataWriter** (p. 1138) was the last to cause this DataReader's status to change, using **dds::sub::DataReader::matched_publication_data** (p. 1033).*
- int32_t **current_count_peak** () const

  *<<**extension**>> (p. 354) Greatest number of DataReaders that matched this **dds::pub::DataWriter** (p. 1138) simultaneously.*

8.370.1 Detailed Description

Information about the status **dds::core::status::StatusMask::subscription_matched()** (p. 2384)

A "match" happens when the **dds::sub::DataReader** (p. 976) finds a **dds::pub::DataWriter** (p. 1138) with the same **dds::topic::Topic** (p. 2474), same or compatible data type, and an offered QoS that is compatible with that requested by the **dds::sub::DataReader** (p. 976). (For information on compatible data types, see the `Extensible Types Guide`.)

This status is also changed (and the listener, if any, called) when a match is ended. A local **dds::sub::DataReader** (p. 976) will become "unmatched" from a remote **dds::pub::DataWriter** (p. 1138) when that **dds::pub::DataWriter** (p. 1138) goes away for any of the following reasons:

- The **dds::domain::DomainParticipant** (p. 1326) containing the matched **dds::pub::DataWriter** (p. 1138) has lost liveliness.
- This DataReader or the matched DataWriter has changed QoS such that the entities are now incompatible.
- The matched DataWriter has been deleted.

This status may reflect changes from multiple match or unmatch events, and the **dds::core::status::SubscriptionMatchedStatus::current_count_change** (p. 2440) can be used to determine the number of changes since the listener was called back or the status was checked.

Note: A DataWriter's loss of liveliness (which is determined by **dds::core::policy::LivelinessKind_def** (p. 1654)) does not trigger an unmatch event. So a DataWriter may still match even though its liveliness is lost.

8.370.2 Member Function Documentation

8.370.2.1 total_count()

```
int32_t dds::core::status::SubscriptionMatchedStatus::total_count ( ) const  [inline]
```

The total cumulative number of times that this **dds::sub::DataReader** (p. 976) discovered a "match" with a **dds::pub↩ ::DataWriter** (p. 1138).

This number increases whenever a new match is discovered. It does not decrease when an existing match goes away for any of the reasons described in **dds::core::status::SubscriptionMatchedStatus** (p. 2439).

8.370.2.2 total_count_change()

```
int32_t dds::core::status::SubscriptionMatchedStatus::total_count_change ( ) const  [inline]
```

The changes in total_count since the last time the listener was called or the status was read.

Note that this number will never be negative (because it's the total number of times the DataReader ever matched with a DataWriter).

8.370.2.3 current_count()

```
int32_t dds::core::status::SubscriptionMatchedStatus::current_count ( ) const  [inline]
```

The current number of DataWriters with which the **dds::sub::DataReader** (p. 976) is matched.

This number increases when a new match is discovered and decreases when an existing match goes away for any of the reasons described in **dds::core::status::SubscriptionMatchedStatus** (p. 2439).

8.370.2.4 current_count_change()

```
int32_t dds::core::status::SubscriptionMatchedStatus::current_count_change ( ) const  [inline]
```

The change in current_count since the last time the listener was called or the status was read.

Note that a negative current_count_change means that one or more DataWriters have become unmatched for one or more of the reasons described in **dds::core::status::SubscriptionMatchedStatus** (p. 2439).

8.370.2.5 last_publication_handle()

```
const dds::core::InstanceHandle dds::core::status::SubscriptionMatchedStatus::last_publication_↩
handle ( ) const [inline]
```

This **InstanceHandle** (p. 1612) can be used to look up which remote **dds::pub::DataWriter** (p. 1138) was the last to cause this DataReader's status to change, using **dds::sub::DataReader::matched_publication_data** (p. 1033).

If the DataWriter no longer matches this DataReader due to any of the reasons in **dds::core::status::Subscription↩ MatchedStatus** (p. 2439) except incompatible QoS, then the DataWriter has been purged from this DataReader's DomainParticipant discovery database. (See the "Discovery Overview" section of the User's Manual.) In that case, the **dds::sub::DataReader::matched_publication_data** (p. 1033) method will not be able to return information about the DataWriter. The only way to get information about the lost DataWriter is if you cached the information previously.

8.370.2.6 current_count_peak()

```
int32_t current_count_peak ( ) const
```

<<***extension***>> *(p. 354)* Greatest number of DataReaders that matched this **dds::pub::DataWriter** (p. 1138) simultaneously.

That is, there was no moment in time when more than this many DataReaders matched this DataWriter. (As a result, total_count can be higher than current_count_peak.)

8.371 dds::pub::SuspendedPublication Class Reference

<<***value-type***>> *(p. 350)* Indicates that the application is about to make multiple modifications using several **dds↩ ::pub::DataWriter** (p. 1138)'s belonging to the same **dds::pub::Publisher** (p. 1993)

```
#include <dds/pub/SuspendedPublication.hpp>
```

Public Member Functions

- **SuspendedPublication** (const **dds::pub::Publisher** &pub)

  Suspends the publications of the publisher.
- void **resume** ()

  Indicates that the application has completed these changes.
- ~**SuspendedPublication** ()

  *Calls **resume()** (p. 2443)*

8.371.1 Detailed Description

<<***value-type***>> *(p. 350)* Indicates that the application is about to make multiple modifications using several **dds::pub::DataWriter** (p. 1138)'s belonging to the same **dds::pub::Publisher** (p. 1993)

It is a **hint** to RTI Connext so it can optimize its performance by e.g., holding the dissemination of the modifications and then batching them.

The use of this operation must be matched by a corresponding call to **dds::pub::SuspendedPublication::resume()** (p. 2443) indicating that the set of modifications has completed.

If the **dds::pub::Publisher** (p. 1993) is deleted before **dds::pub::SuspendedPublication::resume()** (p. 2443) is called, any suspended updates yet to be published will be discarded.

RTI Connext is not required and does not currently make use of this hint in any way. However, similar results can be achieved by using *asynchronous publishing*. Combined with **rti::pub::FlowController** (p. 1572), **rti::core::policy::PublishModeKind_def::ASYNCHRONOUS** (p. 2019) **dds::pub::DataWriter** (p. 1138) instances allow the user even finer control of traffic shaping and sample coalescing.

Exceptions

One	of the **Standard Exceptions** (p. 428) or **dds::core::NotEnabledError** (p. 1869).

See also

rti::pub::FlowController (p. 1572)

rti::pub::FlowController::trigger_flow (p. 1575)

rti::pub::FlowController::ON_DEMAND_NAME (p. 60)

rti::core::policy::PublishMode (p. 2013)

See also

DataWriter Use Cases (p. 311)

Entity Use Cases (p. 325)

8.371.2 Constructor & Destructor Documentation

8.371.2.1 SuspendedPublication()

```
dds::pub::SuspendedPublication::SuspendedPublication (
            const  dds::pub::Publisher & pub )  [inline], [explicit]
```

Suspends the publications of the publisher.

8.371.2.2 ~SuspendedPublication()

```
dds::pub::SuspendedPublication::~SuspendedPublication ( )  [inline]
```

Calls **resume()** (p. 2443)

8.371.3 Member Function Documentation

8.371.3.1 resume()

```
void dds::pub::SuspendedPublication::resume ( )  [inline]
```

Indicates that the application has completed these changes.

References **dds::core::Value**< **D** >**::delegate()**.

8.372 rti::core::policy::SystemResourceLimits Class Reference

<<***extension***>> *(*p. *116)* Configures resources that RTI Connext uses

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **SystemResourceLimits** ()

  Creates a ***SystemResourceLimits*** *(*p. *2443) qos policy with default values.*
- **SystemResourceLimits** (int32_t the_max_objects_per_thread)

  Creates a ***SystemResourceLimits*** *(*p. *2443) qos policy with the provided max_objects_per_thread and set initial_*↩ *objects_per_thread accordingly.*
- **SystemResourceLimits** (int32_t the_max_objects_per_thread, int32_t the_initial_objects_per_thread)

  Creates a ***SystemResourceLimits*** *(*p. *2443) qos policy with the provided max_objects_per_thread and initial_objects*↩ *_per_thread.*
- **SystemResourceLimits** & **max_objects_per_thread** (int32_t the_max_objects_per_thread)

  [DEPRECATED] (p. ***296)*** *Sets the maximum number of objects that can be stored per thread*
- **SystemResourceLimits** & **initial_objects_per_thread** (int32_t the_initial_objects_per_thread)

  Sets the initial number of objects that can be stored per thread.
- int32_t **max_objects_per_thread** () const

  Getter (see setter with the same name)
- int32_t **initial_objects_per_thread** () const

  Getter (see setter with the same name)

8.372.1 Detailed Description

<<***extension***>> *(p. 116)* Configures resources that RTI Connext uses

Entity:

DomainParticipantFactory

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.372.2 Usage

Within a single process (or address space for some supported real-time operating systems), applications may create and use multiple **dds::domain::DomainParticipant** (p. 1326) entities. This QoS policy sets a parameter that places an effective upper bound on the maximum number of **dds::domain::DomainParticipant** (p. 1326) entities that can be created in a single process/address space. These values cannot be changed after a DomainParticipant has been created and they cannot be set in a QoS XML file.

8.372.3 Constructor & Destructor Documentation

8.372.3.1 SystemResourceLimits() [1/3]

```
rti::core::policy::SystemResourceLimits::SystemResourceLimits ( )  [inline]
```

Creates a **SystemResourceLimits** (p. 2443) qos policy with default values.

8.372.3.2 SystemResourceLimits() [2/3]

```
rti::core::policy::SystemResourceLimits::SystemResourceLimits (
            int32_t the_max_objects_per_thread )  [inline], [explicit]
```

Creates a **SystemResourceLimits** (p. 2443) qos policy with the provided max_objects_per_thread and set initial_↩ objects_per_thread accordingly.

8.372.3.3 SystemResourceLimits() [3/3]

```
rti::core::policy::SystemResourceLimits::SystemResourceLimits (
            int32_t the_max_objects_per_thread,
            int32_t the_initial_objects_per_thread )  [inline]
```

Creates a **SystemResourceLimits** (p. 2443) qos policy with the provided max_objects_per_thread and initial_objects↩ _per_thread.

8.372.4 Member Function Documentation

8.372.4.1 max_objects_per_thread() [1/2]

```
SystemResourceLimits & rti::core::policy::SystemResourceLimits::max_objects_per_thread (
            int32_t the_max_objects_per_thread )
```

[DEPRECATED] (p. **296)** Sets the maximum number of objects that can be stored per thread

This value is the upper bound on the number of objects that can be stored per thread. When a DomainParticipantFactory is created, infrastructure will be created to manage the number of objects specified by initial_objects_per_thread. As more objects are required by the application, the infrastructure will be automatically grown to accommodate up to max↩ _objects_per_thread objects. Leave this property set to the default value to allow the infrastructure to grow as needed. If you wish to strictly control memory allocation, set max_objects_per_thread to a smaller value, but note that this runs the risk of a runtime error and reduced application functionality if your limit is reached.

[default] 261120

[range] [1, 261120]

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.372.4.2 initial_objects_per_thread() [1/2]

```
SystemResourceLimits & rti::core::policy::SystemResourceLimits::initial_objects_per_thread (
            int32_t the_initial_objects_per_thread )
```

Sets the initial number of objects that can be stored per thread.

The infrastructure for managing thread-specific objects will initially be sized according to this value. The infrastructure will grow automatically, up to a maximum of max_objects_per_thread, as required by the application at runtime. If you are certain that more than the default value of initial_objects_per_thread will be required for your application and you wish to reduce the number of memory allocations performed while your application reaches steady state, you may set this value to a larger number. To improve the efficiency of memory allocation, RTI Connext may initially size the infrastructure to a larger value than the value of initial_objects_per_thread. The infrastructure will never be sized less than initial_objects_per_thread or greater than max_objects_per_thread. When the infrastructure for managing thread-specific objects is created or increased, a log message stating "Allowed number of thread specific objects is now " will be produced at the local log level.

[default] 1024

[range] [1, 261120]; must be less than or equal to max_objects_per_thread

Categories

Immutable (p. 243), **Middleware Memory Management** (p. 204)

8.372.4.3 max_objects_per_thread() [2/2]

```
int32_t rti::core::policy::SystemResourceLimits::max_objects_per_thread ( ) const
```

Getter (see setter with the same name)

8.372.4.4 initial_objects_per_thread() [2/2]

```
int32_t rti::core::policy::SystemResourceLimits::initial_objects_per_thread ( ) const
```

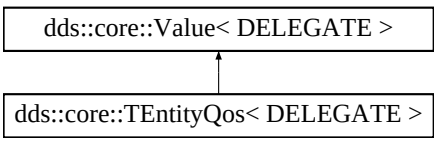
Getter (see setter with the same name)

8.373 dds::core::TEntityQos< DELEGATE > Class Template Reference

Acts as a container for Qos policies allowing to set and retrieve all the policies of an entity as a unit.

```
#include <TEntityQos.hpp>
```

Inheritance diagram for dds::core::TEntityQos< DELEGATE >:



Public Member Functions

- template<typename Policy >
  TEntityQos & **policy** (const Policy &p)

  RTI_doc_block[EntityQosCopy].
- template<typename Policy >
  TEntityQos & **operator<<** (const Policy &p)

  Sets a policy.
- template<typename Policy >
  const **TEntityQos** & **operator>>** (Policy &p) const

  Copies the values of a policy.
- template<typename T >
  TEntityQos< DELEGATE > & **operator=** (const **TEntityQos**< T > &other)

  RTI_doc_block[EntityQosCopy].

8.373.1 Detailed Description

template<typename DELEGATE>
class dds::core::TEntityQos< DELEGATE >

Acts as a container for Qos policies allowing to set and retrieve all the policies of an entity as a unit.

8.373.2 Member Function Documentation

8.373.2.1 policy()

```
template<typename DELEGATE >
template<typename Policy >
TEntityQos &  dds::core::TEntityQos< DELEGATE >::policy (
             const Policy & p )  [inline]
```

RTI_doc_block[EntityQosCopy].

Sets a policy

See also

policy()

Setting Qos Values (p. 603)

8.373.2.2 operator<<()

```
template<typename DELEGATE >
template<typename Policy >
TEntityQos &  dds::core::TEntityQos< DELEGATE >::operator<< (
            const Policy & p )  [inline]
```

Sets a policy.

See also

policy()

Setting Qos Values (p. 603)

8.373.2.3 operator>>()

```
template<typename DELEGATE >
template<typename Policy >
const  TEntityQos &  dds::core::TEntityQos< DELEGATE >::operator>> (
            Policy & p ) const  [inline]
```

Copies the values of a policy.

Parameters

p	The destination where to copy the current value of the Policy

See also

policy()

Setting Qos Values (p. 603)

8.373.2.4 operator=()

```
template<typename DELEGATE >
template<typename T >
TEntityQos< DELEGATE > &  dds::core::TEntityQos< DELEGATE >::operator= (
            const  TEntityQos< T > & other )  [inline]
```

RTI_doc_block[EntityQosCopy].

References **dds::core::Value< DELEGATE >::delegate()**.

8.374 rti::core::ThreadSettings Class Reference

<<***extension***>> *(p. 354)* The properties of a thread of execution.

`#include <rti/core/ThreadSettings.hpp>`

Public Member Functions

- **ThreadSettings** ()

  Creates the default thread settings.
- **ThreadSettings** (**ThreadSettingsKindMask** the_mask, int32_t the_priority, int32_t the_stack_size, **dds::core::vector**< int32_t > the_cpu_list, ThreadSettingsCpuRotationKind::type the_cpu_rotation)

  Creates an instance with all the specified parameters.
- **ThreadSettingsKindMask mask** () const

  Getter (see setter with the same name)
- **ThreadSettings** & **mask** (**ThreadSettingsKindMask** the_mask)

  Describes the type of thread.
- int32_t **priority** () const

  Getter (see setter with the same name)
- **ThreadSettings** & **priority** (int32_t the_priority)

  Thread priority.
- int32_t **stack_size** () const

  Getter (see setter with the same name)
- **ThreadSettings** & **stack_size** (int32_t the_stack_size)

  The thread stack-size.
- **dds::core::vector**< int32_t > **cpu_list** () const

  Getter (see setter with the same name)
- **ThreadSettings** & **cpu_list** (const **dds::core::vector**< int32_t > &the_cpu_list)

  The list of processors on which the thread(s) may run.
- **ThreadSettingsCpuRotationKind cpu_rotation** () const

  Getter (see setter with the same name)
- **ThreadSettings** & **cpu_rotation** (**ThreadSettingsCpuRotationKind** the_cpu_rotation)

  Determines how processor affinity is applied to multiple threads.

8.374.1 Detailed Description

<<***extension***>> *(p. 354)* The properties of a thread of execution.

Consult `Platform Notes` for additional platform specific details.

QoS:

rti::core::policy::Event (p. 1541) **rti::core::policy::Database** (p. 969) **rti::core::policy::ReceiverPool** (p. 2144) **rti::core::policy::AsynchronousPublisher** (p. 832)

8.374.2 Constructor & Destructor Documentation

8.374.2.1 ThreadSettings() [1/2]

```
rti::core::ThreadSettings::ThreadSettings ( )  [inline]
```

Creates the default thread settings.

8.374.2.2 ThreadSettings() [2/2]

```
rti::core::ThreadSettings::ThreadSettings (
             ThreadSettingsKindMask the_mask,
            int32_t the_priority,
            int32_t the_stack_size,
             dds::core::vector< int32_t > the_cpu_list,
            ThreadSettingsCpuRotationKind::type the_cpu_rotation )
```

Creates an instance with all the specified parameters.

8.374.3 Member Function Documentation

8.374.3.1 mask() [1/2]

```
ThreadSettingsKindMask rti::core::ThreadSettings::mask ( ) const
```

Getter (see setter with the same name)

8.374.3.2 mask() [2/2]

```
ThreadSettings & rti::core::ThreadSettings::mask (
             ThreadSettingsKindMask the_mask )
```

Describes the type of thread.

The meaning of each bit of the mask are defined by **rti::core::ThreadSettingsKindMask** (p. 2454).

[default] 0, use default options of the OS

8.374.3.3 priority() [1/2]

```
int32_t rti::core::ThreadSettings::priority ( ) const
```

Getter (see setter with the same name)

8.374.3.4 priority() [2/2]

```
ThreadSettings & rti::core::ThreadSettings::priority (
            int32_t the_priority )
```

Thread priority.

[range] Platform-dependent - Consult `Platform Notes` for additional details.

8.374.3.5 stack_size() [1/2]

```
int32_t rti::core::ThreadSettings::stack_size ( ) const
```

Getter (see setter with the same name)

8.374.3.6 stack_size() [2/2]

```
ThreadSettings & rti::core::ThreadSettings::stack_size (
            int32_t the_stack_size )
```

The thread stack-size.

[range] Platform-dependent. Consult `Platform Notes` for additional details.

Categories

Middleware Memory Management (p. 204)

8.374.3.7 cpu_list() [1/2]

```
dds::core::vector< int32_t > rti::core::ThreadSettings::cpu_list ( ) const
```

Getter (see setter with the same name)

8.374.3.8 cpu_list() [2/2]

```
ThreadSettings & rti::core::ThreadSettings::cpu_list (
            const  dds::core::vector< int32_t > & the_cpu_list )
```

The list of processors on which the thread(s) may run.

A sequence of integers that represent the set of processors on which the thread(s) controlled by this QoS may run. An empty sequence (the default) means the middleware wlll make no CPU affinity adjustments.

Note: This feature is currently only supported on a subset of architectures (see the Platform Notes). The API may change as more architectures are added in future releases.

This value is only relevant to the **rti::core::policy::ReceiverPool** (p. 2144). It is ignored within other QoS policies that include **rti::core::ThreadSettings** (p. 2449).

See also

Controlling CPU Core Affinity for RTI Threads (p. 2453)

[default] Empty sequence

8.374.3.9 cpu_rotation() [1/2]

```
ThreadSettingsCpuRotationKind rti::core::ThreadSettings::cpu_rotation ( ) const
```

Getter (see setter with the same name)

8.374.3.10 cpu_rotation() [2/2]

```
ThreadSettings & rti::core::ThreadSettings::cpu_rotation (
             ThreadSettingsCpuRotationKind the_cpu_rotation )
```

Determines how processor affinity is applied to multiple threads.

This value is only relevant to the **rti::core::policy::ReceiverPool** (p. 2144). It is ignored within other QoS policies that include **rti::core::ThreadSettings** (p. 2449).

See also

Controlling CPU Core Affinity for RTI Threads (p. 2453)

Note: This feature is currently only supported on a subset of architectures (see the Platform Notes). The API may change as more architectures are added in future releases.

8.375 rti::core::ThreadSettingsCpuRotationKind_def Struct Reference

Determines how **rti::core::ThreadSettings::cpu_list** (p. 2451) affects processor affinity for thread-related QoS policies that apply to multiple threads.

`#include <ThreadSettings.hpp>`

Public Types

- enum **type** {
  NO_ROTATION ,
  ROUND_ROBIN }

  The underlying `enum` type.

8.375.1 Detailed Description

Determines how **rti::core::ThreadSettings::cpu_list** (p. 2451) affects processor affinity for thread-related QoS policies that apply to multiple threads.

8.375.2 Controlling CPU Core Affinity for RTI Threads

Most thread-related QoS settings apply to a single thread (such as for the **rti::core::policy::Event** (p. 1541), **rti::core::policy::Database** (p. 969), and **rti::core::policy::AsynchronousPublisher** (p. 832)). However, the thread settings in the **rti::core::policy::ReceiverPool** (p. 2144) control every receive thread created. In this case, there are several schemes to map M threads to N processors; the rotation kind controls which scheme is used.

Controlling CPU Core Affinity is only relevant to the **rti::core::policy::ReceiverPool** (p. 2144). It is ignored within other QoS policies that include **rti::core::ThreadSettings** (p. 2449).

If **rti::core::ThreadSettings::cpu_list** (p. 2451) is empty, the rotation is irrelevant since no affinity adjustment will occur. Suppose instead that **rti::core::ThreadSettings::cpu_list** (p. 2451) = {0, 1} and that the middleware creates three receive threads: {A, B, C}. If **rti::core::ThreadSettings::cpu_rotation** (p. 2452) is **rti::core::ThreadSettingsCpuRotationKind_def::NO_ROTATION** (p. 2454), threads A, B and C will have the same processor affinities (0-1), and the OS will control thread scheduling within this bound. It is common to denote CPU affinities as a bitmask, where set bits represent allowed processors to run on. This mask is printed in hex, so a CPU core affinity of 0-1 can be represented by the mask 0x3.

If **rti::core::ThreadSettings::cpu_rotation** (p. 2452) is **rti::core::ThreadSettingsCpuRotationKind_def::ROUND_ROBIN** (p. 2454), each thread will be assigned in round-robin fashion to one of the processors in **rti::core::ThreadSettings::cpu_list** (p. 2451); perhaps thread A to 0, B to 1, and C to 0. Note that the order in which internal middleware threads spawn is unspecified.

Not all of these options may be relevant for all operating systems. Refer to the Platform Notes for further information.

8.375.3 Member Enumeration Documentation

8.375.3.1 type

`enum` **rti::core::ThreadSettingsCpuRotationKind_def::type**

The underlying `enum` type.

Enumerator

NO_ROTATION	Any thread controlled by this QoS can run on any listed processor, as determined by OS scheduling.
ROUND_ROBIN	Threads controlled by this QoS will be assigned one processor from the list in round-robin order.

8.376 rti::core::ThreadSettingsKindMask Class Reference

<<***extension***>> *(*p. *354)* A collection of flags used to configure threads of execution

`#include <rti/core/ThreadSettings.hpp>`

Inherits std::bitset< 4 >.

Public Types

- typedef std::bitset< 4 > **MaskType**

  The base type, a std::bitset.

Public Member Functions

- **ThreadSettingsKindMask** ()

  The dfault thread settings mask.
- **ThreadSettingsKindMask** (uint64_t mask)

  Create from the bits in an integer.
- **ThreadSettingsKindMask** (const **MaskType** &mask)

  Create from a std::bitset.

Static Public Member Functions

- static const **ThreadSettingsKindMask** **floating_point** ()

  Code executed within the thread may perform floating point operations. Currently applicable only for VxWorks platforms, where user callbacks use floating-point operations.
- static const **ThreadSettingsKindMask** **stdio** ()

  Code executed within the thread may access standard I/O. Currently applicable only for VxWorks platforms, where user callbacks do standard I/O operations.
- static const **ThreadSettingsKindMask** **realtime_priority** ()

  The thread will be scheduled on a FIFO basis.
- static const **ThreadSettingsKindMask** **priority_enforce** ()

  Strictly enforce this thread's priority.
- static const **ThreadSettingsKindMask** **cancel_asynchronous** ()

  Allows the thread to be cancelled without first reaching a cancellable state or cancellation point.

8.376.1 Detailed Description

<<***extension***>> *(p. 354)* A collection of flags used to configure threads of execution

Not all of these options may be relevant for all operating systems. Consult `Platform Notes` for additional details.

8.376.2 Member Typedef Documentation

8.376.2.1 MaskType

```
typedef std::bitset<4>  rti::core::ThreadSettingsKindMask::MaskType
```

The base type, a std::bitset.

8.376.3 Constructor & Destructor Documentation

8.376.3.1 ThreadSettingsKindMask() [1/3]

```
rti::core::ThreadSettingsKindMask::ThreadSettingsKindMask ( )  [inline]
```

The dfault thread settings mask.

8.376.3.2 ThreadSettingsKindMask() [2/3]

```
rti::core::ThreadSettingsKindMask::ThreadSettingsKindMask (
            uint64_t mask )  [inline], [explicit]
```

Create from the bits in an integer.

8.376.3.3 ThreadSettingsKindMask() [3/3]

```
rti::core::ThreadSettingsKindMask::ThreadSettingsKindMask (
            const  MaskType & mask )  [inline]
```

Create from a std::bitset.

8.376.4 Member Function Documentation

8.376.4.1 floating_point()

```
static const ThreadSettingsKindMask rti::core::ThreadSettingsKindMask::floating_point ( ) [inline], [static]
```

Code executed within the thread may perform floating point operations. Currently applicable only for VxWorks platforms, where user callbacks use floating-point operations.

8.376.4.2 stdio()

```
static const ThreadSettingsKindMask rti::core::ThreadSettingsKindMask::stdio ( ) [inline], [static]
```

Code executed within the thread may access standard I/O. Currently applicable only for VxWorks platforms, where user callbacks do standard I/O operations.

8.376.4.3 realtime_priority()

```
static const ThreadSettingsKindMask rti::core::ThreadSettingsKindMask::realtime_priority ( ) [inline], [static]
```

The thread will be scheduled on a FIFO basis.

8.376.4.4 priority_enforce()

```
static const ThreadSettingsKindMask rti::core::ThreadSettingsKindMask::priority_enforce ( ) [inline], [static]
```

Strictly enforce this thread's priority.

8.376.4.5 cancel_asynchronous()

```
static const ThreadSettingsKindMask rti::core::ThreadSettingsKindMask::cancel_asynchronous ( ) [inline], [static]
```

Allows the thread to be cancelled without first reaching a cancellable state or cancellation point.

8.377 dds::core::Time Class Reference

<<***extension***>> *(*p. *354)* Represents a point in time

`#include <Time.hpp>`

Public Member Functions

- **Time** ()

  *Create a default **Time** (*p. *2457) object. The constructed **Time** (*p. *2457) object will represent 0 seconds and 0 nanoseconds.*
- **Time** (int64_t **sec**, uint32_t **nanosec**=0)

  *Create a **Time** (*p. *2457) object. The constructed **Time** (*p. *2457) object will represent the given amount of time.*
- int64_t **sec** () const

  *Get the number of seconds that are represented by this **Time** (*p. *2457) object.*
- void **sec** (int64_t s)

  *Set the number of seconds that are represented by this **Time** (*p. *2457) object.*
- uint32_t **nanosec** () const

  *Get the number of nanoseconds that are represented by this **Time** (*p. *2457) object.*
- void **nanosec** (uint32_t ns)

  *Set the number of nanoseconds that are represented by this **Time** (*p. *2457) object.*
- int **compare** (const **Time** &other) const

  *Compare two **Time** (*p. *2457) objects.*
- bool **operator>** (const **Time** &other) const

  *Check if this **Time** (*p. *2457) is greater than another.*
- bool **operator>=** (const **Time** &other) const

  *Check if this **Time** (*p. *2457) is greater than or equal another.*
- bool **operator==** (const **Time** &other) const

  *Check if this **Time** (*p. *2457) of **Time** (*p. *2457) is equal to another.*
- bool **operator!=** (const **Time** &other) const

  *Check if this **Time** (*p. *2457) is not equal to another.*
- bool **operator<=** (const **Time** &other) const

  *Check if this **Time** (*p. *2457) is less than or equal another.*
- bool **operator<** (const **Time** &other) const

  *Check if this **Time** (*p. *2457) is less than another.*
- **Time** & **operator+=** (const **Duration** &duration)

  *Add a **Duration** (*p. *1448) to this **Time** (*p. *2457) object.*
- **Time** & **operator-=** (const **Duration** &duration)

  *Subtract a **Duration** (*p. *1448) from this **Time** (*p. *2457) object.*
- uint64_t **to_millisecs** () const

  *Convert this **Time** (*p. *2457) to milliseconds.*
- uint64_t **to_microsecs** () const

  *Convert this **Time** (*p. *2457) to microseconds.*
- uint64_t **to_nanosecs** () const

  *Convert this **Time** (*p. *2457) to nanoseconds.*
- double **to_secs** () const

  *Convert this **Time** (*p. *2457) to seconds.*

Static Public Member Functions

- static **Time** **invalid** ()

  *Get a **Time** (p. 2457) object representing an invalid amount of time.*
- static **Time** **zero** ()

  *Get a **Time** (p. 2457) object representing zero time.*
- static **Time** **maximum** ()

  *Get a **Time** (p. 2457) object representing the maximum amount of time.*
- static **Time** **from_microsecs** (uint64_t microseconds)

  *Create a **Time** (p. 2457) object from microseconds.*
- static **Time** **from_millisecs** (uint64_t milliseconds)

  *Create a **Time** (p. 2457) object from milliseconds.*
- static **Time** **from_secs** (double seconds)

  *Create a **Time** (p. 2457) object from seconds.*

Related Functions

(Note that these are not member functions.)

- **Time** **operator+** (const **Time** &time, const **Duration** &duration)

  *Add a **Time** (p. 2457) and a **Duration** (p. 1448) together.*
- **Time** **operator+** (const **Duration** &duration, const **Time** &time)

  *Add a **Time** (p. 2457) and a **Duration** (p. 1448) together.*
- **Time** **operator-** (const **Time** &time, const **Duration** &duration)

  *Subtract a **Duration** (p. 1448) from a **Time** (p. 2457).*
- **Duration** **operator-** (const **Time** &time1, const **Time** &time2)

  Calculate the duration between two times.

8.377.1 Detailed Description

<<***extension***>> *(p. 354)* Represents a point in time

8.377.2 Constructor & Destructor Documentation

8.377.2.1 Time() [1/2]

```
dds::core::Time::Time ( )
```

Create a default **Time** (p. 2457) object. The constructed **Time** (p. 2457) object will represent 0 seconds and 0 nanoseconds.

8.377.2.2 Time() [2/2]

```
dds::core::Time::Time (
            int64_t sec,
            uint32_t nanosec = 0 )  [explicit]
```

Create a **Time** (p. 2457) object. The constructed **Time** (p. 2457) object will represent the given amount of time.

8.377.3 Member Function Documentation

8.377.3.1 invalid()

```
static  Time dds::core::Time::invalid ( )  [static]
```

Get a **Time** (p. 2457) object representing an invalid amount of time.

Returns

const **Time** (p. 2457) A **Time** (p. 2457) object representing an invalid amount of time.

8.377.3.2 zero()

```
static  Time dds::core::Time::zero ( )  [static]
```

Get a **Time** (p. 2457) object representing zero time.

Returns

const **Time** (p. 2457) A **Time** (p. 2457) object representing zero time.

8.377.3.3 maximum()

```
static  Time dds::core::Time::maximum ( )  [static]
```

Get a **Time** (p. 2457) object representing the maximum amount of time.

Returns

const **Time** (p. 2457) A **Time** (p. 2457) object representing the maximum amount of time.

8.377.3.4 from_microsecs()

```
static  Time dds::core::Time::from_microsecs (
            uint64_t microseconds )  [static]
```

Create a **Time** (p. 2457) object from microseconds.

Parameters

microseconds	How many microseconds this **Time** (p. 2457) represent

Returns

Time (p. 2457) A new instance of **Time** (p. 2457) representing the given microseconds

8.377.3.5 from_millisecs()

```
static  Time dds::core::Time::from_millisecs (
             uint64_t milliseconds )  [static]
```

Create a **Time** (p. 2457) object from milliseconds.

Parameters

milliseconds	How many milliseconds this **Time** (p. 2457) should represent

Returns

Time (p. 2457) A new instance of **Time** (p. 2457) representing the given milliseconds

8.377.3.6 from_secs()

```
static  Time dds::core::Time::from_secs (
             double seconds )  [static]
```

Create a **Time** (p. 2457) object from seconds.

Parameters

seconds	How many seconds this **Time** (p. 2457) should represent

Returns

Time (p. 2457) A new instance of **Time** (p. 2457) representing the given seconds

8.377.3.7 sec() [1/2]

```
int64_t dds::core::Time::sec ( ) const
```

Get the number of seconds that are represented by this **Time** (p. 2457) object.

Returns

int64_t The number of seconds

8.377.3.8 sec() [2/2]

```
void dds::core::Time::sec (
            int64_t s )
```

Set the number of seconds that are represented by this **Time** (p. 2457) object.

Parameters

s	The number of seconds to set.

8.377.3.9 nanosec() [1/2]

```
uint32_t dds::core::Time::nanosec ( ) const
```

Get the number of nanoseconds that are represented by this **Time** (p. 2457) object.

Returns

uint32_t The number of nanoseconds

8.377.3.10 nanosec() [2/2]

```
void dds::core::Time::nanosec (
            uint32_t ns )
```

Set the number of nanoseconds that are represented by this **Time** (p. 2457) object.

Parameters

ns	The number of nanoseconds to set.

8.377.3.11 compare()

```
int dds::core::Time::compare (
            const  Time & other ) const
```

Compare two **Time** (p. 2457) objects.

Parameters

other	The **Time** (p. 2457) object to compare with this **Time** (p. 2457).

Returns

int The result of the comparison can be: -1 if this **Time** (p. 2457) is less than other; 0 if they are equal; 1 if this **Time** (p. 2457) is greater than other

8.377.3.12 operator>()

```
bool dds::core::Time::operator> (
            const  Time & other ) const
```

Check if this **Time** (p. 2457) is greater than another.

Parameters

other	The **Time** (p. 2457) object to compare with this **Time** (p. 2457).

Returns

bool false if this **Time** (p. 2457) is less than or equal to the other object, true otherwise.

8.377.3.13 operator>=()

```
bool dds::core::Time::operator>= (
            const  Time & other ) const
```

Check if this **Time** (p. 2457) is greater than or equal another.

Parameters

other	The **Time** (p. 2457) object to compare with this **Time** (p. 2457).

Returns

bool false if this **Time** (p. 2457) is less than to the other object, true otherwise.

8.377.3.14 operator==()

```
bool dds::core::Time::operator== (
             const  Time & other ) const
```

Check if this **Time** (p. 2457) of **Time** (p. 2457) is equal to another.

Parameters

other	The **Time** (p. 2457) object to compare with this **Time** (p. 2457).

Returns

bool false if this **Time** (p. 2457) is not equal to the other object, true otherwise.

8.377.3.15 operator"!=()

```
bool dds::core::Time::operator!= (
             const  Time & other ) const
```

Check if this **Time** (p. 2457) is not equal to another.

Parameters

other	The **Time** (p. 2457) object to compare with this **Time** (p. 2457).

Returns

bool false if this **Time** (p. 2457) is equal to the other object, true otherwise.

8.377.3.16 operator<=()

```
bool dds::core::Time::operator<= (
            const  Time & other ) const
```

Check if this **Time** (p. 2457) is less than or equal another.

Parameters

other	The **Time** (p. 2457) object to compare with this **Time** (p. 2457).

Returns

bool false if this **Time** (p. 2457) is greater than to the other object, true otherwise.

8.377.3.17 operator<()

```
bool dds::core::Time::operator< (
            const  Time & other ) const
```

Check if this **Time** (p. 2457) is less than another.

Parameters

other	The **Time** (p. 2457) object to compare with this **Time** (p. 2457).

Returns

bool false if this **Time** (p. 2457) is greater than or equal to the other object, true otherwise.

8.377.3.18 operator+=()

```
Time & dds::core::Time::operator+= (
            const  Duration & duration )
```

Add a **Duration** (p. 1448) to this **Time** (p. 2457) object.

Parameters

duration	The **Duration** (p. 1448) to add

Returns

Time (p. 2457)& This **Time** (p. 2457), with the added **Duration** (p. 1448)

8.377.3.19 operator-=()

```
Time & dds::core::Time::operator-= (
            const  Duration & duration )
```

Subtract a **Duration** (p. 1448) from this **Time** (p. 2457) object.

Parameters

duration	The **Duration** (p. 1448) to subtract

Returns

Time (p. 2457)& This **Time** (p. 2457), with the subtracted **Duration** (p. 1448)

8.377.3.20 to_millisecs()

```
uint64_t dds::core::Time::to_millisecs ( ) const
```

Convert this **Time** (p. 2457) to milliseconds.

Returns

uint64_t The number of milliseconds represented by this **Time** (p. 2457) object.

Exceptions

std::overflow_error	if the time in milliseconds exceeds the value that can be stored in an uint64_t

8.377.3.21 to_microsecs()

```
uint64_t dds::core::Time::to_microsecs ( ) const
```

Convert this **Time** (p. 2457) to microseconds.

Returns

uint64_t The number of microseconds represented by this **Time** (p. 2457) object.

Exceptions

std::overflow_error	if the time in microseconds exceeds the value that can be stored in an uint64_t

8.377.3.22 to_nanosecs()

```
uint64_t dds::core::Time::to_nanosecs ( ) const
```

Convert this **Time** (p. 2457) to nanoseconds.

Returns

uint64_t The number of nanoseconds represented by this **Time** (p. 2457) object.

Exceptions

std::overflow_error	if the time in nanoseconds exceeds the value that can be stored in an uint64_t

8.377.3.23 to_secs()

```
double dds::core::Time::to_secs ( ) const
```

Convert this **Time** (p. 2457) to seconds.

Returns

double The number of seconds represented by this **Time** (p. 2457) object.

8.377.4 Friends And Related Function Documentation

8.377.4.1 operator+() [1/2]

```
Time operator+ (
            const  Time & time,
            const  Duration & duration )  [related]
```

Add a **Time** (p. 2457) and a **Duration** (p. 1448) together.

Parameters

time	The **Time** (p. 2457) object to add
duration	The **Duration** (p. 1448) object to add

Returns

Time (p. 2457) The result of the addition represented as a **Time** (p. 2457) object

8.377.4.2 operator+() [2/2]

```
Time operator+ (
            const  Duration & duration,
            const  Time & time )  [related]
```

Add a **Time** (p. 2457) and a **Duration** (p. 1448) together.

Parameters

duration	The **Duration** (p. 1448) object to add
time	The **Time** (p. 2457) object to add

Returns

Time (p. 2457) The result of the addition represented as a **Time** (p. 2457) object

8.377.4.3 operator-() [1/2]

```
Time operator- (
            const  Time & time,
            const  Duration & duration )  [related]
```

Subtract a **Duration** (p. 1448) from a **Time** (p. 2457).

Parameters

time	The **Time** (p. 2457) object to subtract from
duration	The **Duration** (p. 1448) object to subtract

Returns

Time (p. 2457) The result of the subtraction represented as a **Time** (p. 2457) object

8.377.4.4 operator-() [2/2]

```
Duration operator- (
            const  Time & time1,
            const  Time & time2 )  [related]
```

Calculate the duration between two times.

Parameters

time1	The first ("before") time
time2	The second ("after") time

Exceptions

std::overflow_error	if duration exceeds the infinite limit

Returns

The duration elapsed between time1 and time2. If time1 $>$ time2 the duration is zero. If time1 is **Time::maximum()** (p. 2459), the duration is **Duration::infinite()** (p. 1451);

8.378 dds::core::policy::TimeBasedFilter Class Reference

Allows a **dds::sub::DataReader** (p. 976) to indicate that it is not interested in all the sample updates that occur within a time period.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **TimeBasedFilter** ()

  Creates the default time-based filter.
- **TimeBasedFilter** (const **dds::core::Duration** &the_min_separation)

  Creates a policy with the specified minimum separation.
- **TimeBasedFilter** & **minimum_separation** (const **dds::core::Duration** &min_separation)

  Sets the minimum separation between subsequent samples.
- const **dds::core::Duration minimum_separation** () const

  Getter (see setter with the same name)

8.378.1 Detailed Description

Allows a **dds::sub::DataReader** (p. 976) to indicate that it is not interested in all the sample updates that occur within a time period.

The filter states that the **dds::sub::DataReader** (p. 976) does not want to receive more than one value each `minimum_separation`, regardless of how fast the changes occur.

Entity:

dds::sub::DataReader (p. 976)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **YES** (p. 514)

8.378.2 Usage

You can use this QoS policy to reduce the amount of data received by a **dds::sub::DataReader** (p. 976). **dds::pub::DataWriter** (p. 1138) entities may send data faster than needed by a **dds::sub::DataReader** (p. 976). For example, a **dds::sub::DataReader** (p. 976) of sensor data that is displayed to a human operator in a GUI application does not need to receive data updates faster than a user can reasonably perceive changes in data values. This is often measured in tenths (0.1) of a second up to several seconds. However, a **dds::pub::DataWriter** (p. 1138) of sensor information may have other **dds::sub::DataReader** (p. 976) entities that are processing the sensor information to control parts of the system and thus need new data updates in measures of hundredths (0.01) or thousandths (0.001) of a second.

With this QoS policy, different **dds::sub::DataReader** (p. 976) entities can set their own time-based filters, so that data published faster than the period set by a each **dds::sub::DataReader** (p. 976) will not be delivered to that **dds::sub::DataReader** (p. 976).

The **TIME_BASED_FILTER** (p. 547) also applies to each instance separately; that is, the constraint is that the **dds::sub::DataReader** (p. 976) does not want to see more than one sample of each instance per `minimum_separation` period.

This QoS policy allows you to optimize resource usage (CPU and possibly network bandwidth) by only delivering the required amount of data to each **dds::sub::DataReader** (p. 976), accommodating the fact that, for rapidly-changing data, different subscribers may have different requirements and constraints as to how frequently they need or can handle being notified of the most current values. As such, it can also be used to protect applications that are running on a heterogeneous network where some nodes are capable of generating data much faster than others can consume it.

For best effort data delivery, if the data type is unkeyed and the **dds::pub::DataWriter** (p. 1138) has an infinite **dds::core::policy::Liveliness::lease_duration** (p. 1649), RTI Connext will only send as many packets to a **dds::sub::DataReader** (p. 976) as required by the TIME_BASED_FILTER, no matter how fast **dds::pub::DataWriter::write()** (p. 1148) is called.

For multicast data delivery to multiple DataReaders, the one with the lowest `minimum_separation` determines the DataWriter's send rate. For example, if a **dds::pub::DataWriter** (p. 1138) sends over multicast to two DataReaders, one with `minimum_separation` of 2 seconds and one with `minimum_separation` of 1 second, the DataWriter will send every 1 second.

In configurations where RTI Connext must send all the data published by the **dds::pub::DataWriter** (p. 1138) (for example, when the **dds::pub::DataWriter** (p. 1138) is reliable, when the data type is keyed, or when the **dds::pub::DataWriter** (p. 1138) has a finite **dds::core::policy::Liveliness::lease_duration** (p. 1649)), only the data that passes the TIME_BASED_FILTER will be stored in the receive queue of the **dds::sub::DataReader** (p. 976). Extra data will be accepted but dropped. Note that filtering is only applied on alive samples (that is, samples that have not been disposed/unregistered).

8.378.3 Consistency

It is inconsistent for a **dds::sub::DataReader** (p. 976) to have a `minimum_separation` longer than its **DEADLINE** (p. 523) `period`.

However, it is important to be aware of certain edge cases that can occur when your publication rate, minimum separation, and deadline period align and that can cause missed deadlines that you may not expect. For example, suppose that you nominally publish samples every second but that this rate can vary somewhat over time. You declare a minimum separation of 1 second to filter out rapid updates and set a deadline of two seconds so that you will be aware if the rate falls too low. Even if your update rate never wavers, you can still miss deadlines! Here's why:

Suppose you publish the first sample at time t=0 seconds. You then publish your next sample at t=1 seconds. Depending on how your operating system schedules the time-based filter execution relative to the publication, this second sample may be filtered. You then publish your third sample at t=2 seconds, and depending on how your OS schedules this publication in relation to the deadline check, you could miss the deadline.

This scenario demonstrates a couple of rules of thumb:

- Beware of setting your `minimum_separation` to a value very close to your publication rate: you may filter more data than you intend to.
- Beware of setting your `minimum_separation` to a value that is too close to your deadline period relative to your publication rate. You may miss deadlines.

See **dds::core::policy::Deadline** (p. 1255) for more information about the interactions between deadlines and time-based filters.

The setting of a **TIME_BASED_FILTER** (p. 547) – that is, the selection of a `minimum_separation` with a value greater than zero – is consistent with all settings of the **HISTORY** (p. 533) and **RELIABILITY** (p. 544) QoS. The **TIME_↩ BASED_FILTER** (p. 547) specifies the samples that are of interest to the **dds::sub::DataReader** (p. 976). The **HISTORY** (p. 533) and **RELIABILITY** (p. 544) QoS affect the behavior of the middleware with respect to the samples that have been determined to be of interest to the **dds::sub::DataReader** (p. 976); that is, they apply *after* the **TIME_BASED_FILTER** (p. 547) has been applied.

In the case where the reliability QoS kind is **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157), in steady-state – defined as the situation where the **dds::pub::DataWriter** (p. 1138) does not write new samples for a period "long" compared to the `minimum_separation` – the system should guarantee delivery of the last sample to the **dds::sub::DataReader** (p. 976).

See also

DeadlineQosPolicy

HistoryQosPolicy

ReliabilityQosPolicy

8.378.4 Constructor & Destructor Documentation

8.378.4.1 TimeBasedFilter() [1/2]

```
dds::core::policy::TimeBasedFilter::TimeBasedFilter ( )  [inline]
```

Creates the default time-based filter.

8.378.4.2 TimeBasedFilter() [2/2]

```
dds::core::policy::TimeBasedFilter::TimeBasedFilter (
            const  dds::core::Duration & the_min_separation )  [inline], [explicit]
```

Creates a policy with the specified minimum separation.

8.378.5 Member Function Documentation

8.378.5.1 minimum_separation() [1/2]

```
TimeBasedFilter & dds::core::policy::TimeBasedFilter::minimum_separation (
            const  dds::core::Duration & min_separation )  [inline]
```

Sets the minimum separation between subsequent samples.

[default] 0 (meaning the **dds::sub::DataReader** (p. 976) is potentially interested in all values)

[range] [0,1 year], $<$ **dds::core::policy::Deadline::period** (p. 1258)

Categories

Mutable (p. 280), **Discovery Metadata** (p. 288), **Applied Per-Instance** (p. 287), **Data Availability** (p. 167)

8.378.5.2 minimum_separation() [2/2]

```
const  dds::core::Duration dds::core::policy::TimeBasedFilter::minimum_separation ( ) const  [inline]
```

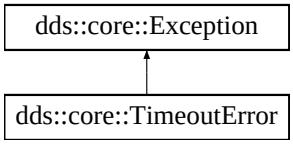
Getter (see setter with the same name)

8.379 dds::core::TimeoutError Class Reference

Indicates that an operation has timed out.

`#include <Exception.hpp>`

Inheritance diagram for dds::core::TimeoutError:



Public Member Functions

- virtual const char * **what** () const throw ()

  *Access the message contained in this **TimeoutError** (p. 2473) exception.*

8.379.1 Detailed Description

Indicates that an operation has timed out.

Inherits also from `std::runtime_error`

8.379.2 Member Function Documentation

8.379.2.1 what()

```
virtual const char * dds::core::TimeoutError::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **TimeoutError** (p. 2473) exception.

Returns

The message.

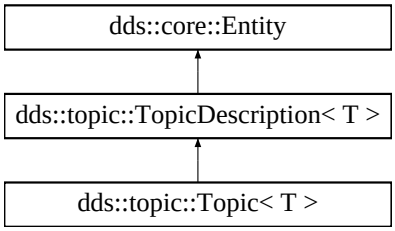
Implements **dds::core::Exception** (p. 1547).

8.380 dds::topic::Topic< T > Class Template Reference

<<***reference-type***>> *(p. 351)* **Topic** (p. 2474) is the most basic description of the data to be published and subscribed.

```
#include <dds/topic/Topic.hpp>
```

Inheritance diagram for dds::topic::Topic< T >:



Public Member Functions

- **Topic** (const **dds::domain::DomainParticipant** & **participant**, RTI_STRING_IN topic_name)

  Creates a new topic.
- **Topic** (const **dds::domain::DomainParticipant** &dp, RTI_STRING_IN topic_name, RTI_STRING_IN **type_name**)

  Creates a new topic.
- **Topic** (const **dds::domain::DomainParticipant** &dp, RTI_STRING_IN topic_name, const **dds::topic::qos::TopicQos** & **qos**, **dds::topic::TopicListener**< T > * **listener**=NULL, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Creates a new **Topic** (p. 2474).*
- **Topic** (const **dds::domain::DomainParticipant** & **participant**, RTI_STRING_IN topic_name, RTI_STRING_IN **type_name**, const **dds::topic::qos::TopicQos** & **qos**, **dds::topic::TopicListener**< T > * **listener**=NULL, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  Create a new topic.
- **Topic** (const **dds::domain::DomainParticipant** & **participant**, RTI_STRING_IN topic_name, const **dds::core::xtypes::DynamicType** &type)

  Create a new topic with a dynamic type description.
- **Topic** (const **dds::domain::DomainParticipant** & **participant**, RTI_STRING_IN topic_name, const **dds::core::xtypes::DynamicType** &type, const **dds::topic::qos::TopicQos** & **qos**, **dds::topic::TopicListener**< T > * **listener**=NULL, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  Create a new topic with a dynamic type description.
- **Topic** (const **dds::domain::DomainParticipant** &dp, RTI_STRING_IN topic_name, const **dds::topic::qos::TopicQos** & **qos**, std::shared_ptr< **Listener** > **listener**, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  *Creates a new **Topic** (p. 2474).*
- **Topic** (const **dds::domain::DomainParticipant** & **participant**, RTI_STRING_IN topic_name, RTI_STRING_IN **type_name**, const **dds::topic::qos::TopicQos** & **qos**, std::shared_ptr< **Listener** > **listener**, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

  Create a new topic.
- **Topic** (const **dds::domain::DomainParticipant** & **participant**, RTI_STRING_IN topic_name, const **dds::core::xtypes::DynamicType** &type, const **dds::topic::qos::TopicQos** & **qos**, std::shared_ptr< **Listener** > **listener**, const **dds::core::status::StatusMask** &mask= **dds::core::status::StatusMask::all**())

Create a new topic with a dynamic type description.

- void **listener** (**Listener** *listener, const **::dds::core::status::StatusMask** &event_mask)

  Sets the listener.
- **Listener** * **listener** () const

  Gets the listener.
- void **set_listener** (std::shared_ptr< **Listener** > **listener**, const **dds::core::status::StatusMask** &event_mask)

  Sets the listener associated with this topic.
- void **set_listener** (std::shared_ptr< **Listener** > **listener**)

  Sets the listener associated with this topic.
- std::shared_ptr< **Listener** > **get_listener** () const

  Returns the listener currently associated with this topic.
- const **dds::topic::qos::TopicQos** **qos** () const

  *Gets the current TopicQos for this **Topic*** (p. 2474).
- void **qos** (const **dds::topic::qos::TopicQos** &qos)

  *Sets the TopicQos setting for this **Topic*** (p. 2474).
- **dds::core::status::InconsistentTopicStatus** **inconsistent_topic_status** () const

  *Retrieve the current InconsistentTopicStatus of this **Topic*** (p. 2474).

Related Functions

(Note that these are not member functions.)

- template<typename FwdIterator >
  int32_t **discover_any_topic** (const **dds::domain::DomainParticipant** & **participant**, FwdIterator begin, int32_t max_size)

  Retrieves the topics discovered in a DomainParticipant.
- template<typename BinIterator >
  int32_t **discover_any_topic** (const **dds::domain::DomainParticipant** & **participant**, BinIterator begin)

  Retrieves the topics discovered in a DomainParticipant.
- **dds::topic::TopicBuiltinTopicData** **discover_topic_data** (const **dds::domain::DomainParticipant** & **participant**, const **dds::core::InstanceHandle** &topic_handle)

  Gets the information about a discovered topic.
- template<typename FwdIterator >
  int32_t **discover_topic_data** (const **dds::domain::DomainParticipant** & **participant**, FwdIterator begin, const **dds::core::InstanceHandleSeq** &handles)

  Gets the information about several topics.
- template<typename FwdIterator >
  int32_t **discover_topic_data** (const **dds::domain::DomainParticipant** & **participant**, FwdIterator begin, int32_t max_size)

  Gets the information about all topics.
- template<typename BinIterator >
  int32_t **discover_topic_data** (const **dds::domain::DomainParticipant** & **participant**, BinIterator begin)

  Gets the information about all topics.
- void **ignore** (**dds::domain::DomainParticipant** & **participant**, const **dds::core::InstanceHandle** &handle)

  Instructs a DomainParticipant to ignore a topic.
- template<typename FwdIterator >
  void **ignore** (**dds::domain::DomainParticipant** & **participant**, FwdIterator begin, FwdIterator end)

Ignores a range of topics.

- template<typename TOPIC >
  TOPIC **find** (const **dds::domain::DomainParticipant** & **participant**, const std::string &topic_name)

  *Looks up a **Topic** (p. 2474) from a DomainParticipant.*
- template<typename AnyTopicBackInsertIterator >
  uint32_t **find_topics** (**dds::domain::DomainParticipant participant**, AnyTopicBackInsertIterator begin)

  <<***extension***>> *(p. 354) Retrieve all the **dds::topic::Topic** (p. 2474) created from this **dds::domain::DomainParticipant** (p. 1326)*
- template<typename AnyTopicForwardIterator >
  uint32_t **find_topics** (**dds::domain::DomainParticipant participant**, AnyTopicForwardIterator begin, uint32_t max_size)

  <<***extension***>> *(p. 354) Retrieve all the **dds::topic::Topic** (p. 2474) created from this **dds::domain::DomainParticipant** (p. 1326)*

8.380.1 Detailed Description

template<typename T>
class dds::topic::Topic< T >

<<***reference-type***>> *(p. 351)* **Topic** (p. 2474) is the most basic description of the data to be published and subscribed.

Template Parameters

T	The type associated to this topic. It must be a type such that `dds::topic::is_topic_type<T>::value` is true (see **dds::topic::is_topic_type** (p. 1622)). Valid types are **IDL-generated types** (p. 606), the **built-in types** (p. 50) and **dds::core::xtypes::DynamicData** (p. 1461).

QoS:

dds::topic::qos::TopicQos (p. 2509)

Status:

dds::core::status::StatusMask::inconsistent_topic() (p. 2379), **dds::core::status::InconsistentTopicStatus** (p. 1611)

Listener:

TopicListener (p. 2508)

A **dds::topic::Topic** (p. 2474) is identified by its name, which must be unique in the whole domain. In addition (by virtue of extending **TopicDescription** (p. 2503)) it fully specifies the type of the data that can be communicated when publishing or subscribing to the **dds::topic::Topic** (p. 2474).

dds::topic::Topic (p. 2474) is the only **TopicDescription** (p. 2503) that can be used for publications and therefore associated with a **dds::pub::DataWriter** (p. 1138).

The following operations may be called even if the **dds::topic::Topic** (p. 2474) is not enabled. Other operations will fail with the value **dds::core::NotEnabledError** (p. 1869) if called on a disabled **dds::topic::Topic** (p. 2474):

- **dds::core::Entity::enable** (p. 1524)
- **dds::topic::Topic::qos(const dds::topic::qos::TopicQos&)** (p. 2484), **dds::topic::Topic::qos() const** (p. 2483)
- **dds::topic::Topic::set_listener** (p. 2482), **dds::topic::Topic::get_listener** (p. 2483)
- **dds::core::cond::StatusCondition::StatusCondition(const dds::core::Entity&)** (p. 2373), **dds::core::Entity::status_changes()** (p. 1525)
- **dds::topic::Topic::inconsistent_topic_status** (p. 2484)

See also

Operations Allowed in Listener Callbacks (p. 1639)

Examples

Foo_publisher.cxx, and **Foo_subscriber.cxx**.

8.380.2 Constructor & Destructor Documentation

8.380.2.1 Topic() [1/9]

```
template<typename T >
dds::topic::Topic< T >::Topic (
            const  dds::domain::DomainParticipant & participant,
            RTI_STRING_IN topic_name )  [inline]
```

Creates a new topic.

This constructor automatically deduces the type name, uses **the default TopicQos** (p. 1522) and doesn't set a listener.

Parameters

participant	The domain participant on which the topic will be defined.
topic_name	The topic name.

8.380.2.2 Topic() [2/9]

```
template<typename T >
dds::topic::Topic< T >::Topic (
            const  dds::domain::DomainParticipant & dp,
```

```
            RTI_STRING_IN topic_name,
            RTI_STRING_IN type_name )  [inline]
```

Creates a new topic.

This constructor uses **the default TopicQos** (p. 1522) and doesn't set a listener.

Parameters

dp	The domain participant on which the topic will be defined.
topic_name	The topic name.
type_name	The name to register the type with

8.380.2.3 Topic() [3/9]

```
template<typename T >
dds::topic::Topic< T >::Topic (
            const  dds::domain::DomainParticipant & dp,
            RTI_STRING_IN topic_name,
            const  dds::topic::qos::TopicQos & qos,
             dds::topic::TopicListener< T > * listener = NULL,
            const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Creates a new **Topic** (p. 2474).

[DEPRECATED] When using a listener, prefer the constructor that receives a shared_ptr<**Listener** (p. 1637)> instead of a regular Listener* pointer.

This constructor deduces the type name automatically

8.380.2.4 Topic() [4/9]

```
template<typename T >
dds::topic::Topic< T >::Topic (
            const  dds::domain::DomainParticipant & participant,
            RTI_STRING_IN topic_name,
            RTI_STRING_IN type_name,
            const  dds::topic::qos::TopicQos & qos,
             dds::topic::TopicListener< T > * listener = NULL,
            const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a new topic.

[DEPRECATED] When using a listener, prefer the constructor that receives a shared_ptr<**Listener** (p. 1637)> instead of a regular Listener* pointer.

Parameters

participant	The domain participant on which the topic will be defined.
topic_name	The **Topic** (p. 2474) name. Must not exceed 255 characters.
type_name	The name given to the type `T`
qos	The Qos settings for this **Topic** (p. 2474)
listener	the topic listener (default: NULL)
mask	the event mask associated to the DataReader listener.

See also

Specifying QoS on entities (p. 513) for information on setting QoS before entity creation

dds::topic::qos::TopicQos (p. 2509) for rules on consistency among QoS

dds::core::QosProvider (p. 2025)

dds::domain::DomainParticipant::default_topic_qos() (p. 1344)

listener() (p. 2482)

8.380.2.5 Topic() [5/9]

```
template<typename T >
dds::topic::Topic< T >::Topic (
             const  dds::domain::DomainParticipant & participant,
             RTI_STRING_IN topic_name,
             const  dds::core::xtypes::DynamicType & type )  [inline]
```

Create a new topic with a dynamic type description.

This constructor uses a dynamic type description, **the default TopicQos** (p. 1522) and doesn't set a listener.

8.380.2.6 Topic() [6/9]

```
template<typename T >
dds::topic::Topic< T >::Topic (
             const  dds::domain::DomainParticipant & participant,
             RTI_STRING_IN topic_name,
             const  dds::core::xtypes::DynamicType & type,
             const  dds::topic::qos::TopicQos & qos,
              dds::topic::TopicListener< T > * listener = NULL,
             const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a new topic with a dynamic type description.

[DEPRECATED] When using a listener, prefer the constructor that receives a `shared_ptr<`**Listener** (p. 1637)`>` instead of a regular `Listener*` pointer.

Notice that in this case the data type `T` has to be DynamicData. Thus the **Topic** (p. 2474) type will be `Topic<`↩ `DynamicData>`.

Parameters

participant	the domain participant on which the topic will be defined.
topic_name	the topic's name.
type	The DynamicType that describes the type for the DynamicData samples of this topic.
qos	The Qos settings for this **Topic** (p. 2474)
listener	the topic listener (default: NULL)
mask	the event mask associated to the DataReader listener.

See also

dds::core::xtypes::DynamicData (p. 1461)

rti::domain::register_type() (p. 733)

8.380.2.7 Topic() [7/9]

```
template<typename T >
dds::topic::Topic< T >::Topic (
             const  dds::domain::DomainParticipant & dp,
             RTI_STRING_IN topic_name,
             const  dds::topic::qos::TopicQos & qos,
             std::shared_ptr<  Listener > listener,
             const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Creates a new **Topic** (p. 2474).

This constructor deduces the type name automatically.

8.380.2.8 Topic() [8/9]

```
template<typename T >
dds::topic::Topic< T >::Topic (
             const  dds::domain::DomainParticipant & participant,
             RTI_STRING_IN topic_name,
             RTI_STRING_IN type_name,
             const  dds::topic::qos::TopicQos & qos,
             std::shared_ptr<  Listener > listener,
             const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a new topic.

Parameters

participant	The domain participant on which the topic will be defined.
topic_name	The **Topic** (p. 2474) name. Must not exceed 255 characters.
type_name	The name given to the type `T`
qos	The Qos settings for this **Topic** (p. 2474)
listener	A shared_ptr to the listener. See **set_listener()** (p. 2482) for more information.
mask	Indicates which status updates the listener will receive

See also

Specifying QoS on entities (p. 513) for information on setting QoS before entity creation

dds::topic::qos::TopicQos (p. 2509) for rules on consistency among QoS

dds::core::QosProvider (p. 2025)

dds::domain::DomainParticipant::default_topic_qos() (p. 1344)

listener() (p. 2482)

8.380.2.9 Topic() [9/9]

```
template<typename T >
dds::topic::Topic< T >::Topic (
            const  dds::domain::DomainParticipant & participant,
            RTI_STRING_IN topic_name,
            const  dds::core::xtypes::DynamicType & type,
            const  dds::topic::qos::TopicQos & qos,
            std::shared_ptr<  Listener > listener,
            const  dds::core::status::StatusMask & mask =  dds::core::status::StatusMask::all() )
[inline]
```

Create a new topic with a dynamic type description.

Notice that in this case the data type `T` has to be DynamicData. Thus the **Topic** (p. 2474) type will be `Topic<DynamicData>`.

Parameters

participant	the domain participant on which the topic will be defined.
topic_name	the topic's name.
type	The DynamicType that describes the type for the DynamicData samples of this topic.
qos	The Qos settings for this **Topic** (p. 2474)
listener	A shared_ptr to the listener. See **set_listener()** (p. 2482) for more information.
mask	Indicates which status updates the listener will receive

See also

dds::core::xtypes::DynamicData (p. 1461)

rti::domain::register_type() (p. 733)

8.380.3 Member Function Documentation

8.380.3.1 listener() [1/2]

```
template<typename T >
void  dds::topic::Topic< T >::listener (
             Listener * listener,
            const  ::dds::core::status::StatusMask & event_mask )  [inline]
```

Sets the listener.

[DEPRECATED] The use of **set_listener()** (p. 2482) is recommended. Unlike this function, set_listener receives a shared_ptr which simplifies the management of listener's lifecycle.

Parameters

listener	the topic listener
event_mask	Changes of communication status to be invoked on the listener

8.380.3.2 listener() [2/2]

```
template<typename T >
Listener *  dds::topic::Topic< T >::listener ( ) const  [inline]
```

Gets the listener.

[DEPRECATED] Prefer **get_listener()** (p. 2483) instead of this function.

Returns the listener or NULL if there's no listener attached.

8.380.3.3 set_listener() [1/2]

```
template<typename T >
void  dds::topic::Topic< T >::set_listener (
            std::shared_ptr<  Listener > listener,
            const  dds::core::status::StatusMask & event_mask )  [inline]
```

Sets the listener associated with this topic.

The topic will hold a shared_ptr to the listener argument, ensuring that it is not deleted at least until this topic is deleted or the listener is reset with set_listener(nullptr).

Warning

It's recommended that the listener implementation doesn't hold a permanent reference to the topic. If it does, the application needs to manually reset the listener or manually close this topic to ensure that there is no cycle that prevents the destruction of these two objects.

Parameters

listener	A shared pointer to the listener to receive status updates or `nullptr` to reset the current listener and stop receiving status updates.
event_mask	A mask that indicates which status updates will be notified to the listener

8.380.3.4 set_listener() [2/2]

```
template<typename T >
void  dds::topic::Topic< T >::set_listener (
            std::shared_ptr<  Listener > listener )  [inline]
```

Sets the listener associated with this topic.

If `listener` is not `nullptr`, this overload is equivalent to:
```
topic.set_listener(listener, dds::core::status::StatusMask::all());
```

If `listener` is `nullptr`, it is equivalent to:
```
topic.set_listener(nullptr, dds::core::status::StatusMask::none());
```

Parameters

listener	A shared pointer to the listener to receive status updates or `nullptr` to reset the current listener and stop receiving status updates.

8.380.3.5 get_listener()

```
template<typename T >
std::shared_ptr<  Listener >  dds::topic::Topic< T >::get_listener ( ) const  [inline]
```

Returns the listener currently associated with this topic.

Returns

The shared pointer to the current listener or `nullptr` if there is currently no listener associated to this entity.

8.380.3.6 qos() [1/2]

```
template<typename T >
const  dds::topic::qos::TopicQos  dds::topic::Topic< T >::qos ( ) const  [inline]
```

Gets the current TopicQos for this **Topic** (p. 2474).

8.380.3.7 qos() [2/2]

```
template<typename T >
void  dds::topic::Topic< T >::qos (
            const  dds::topic::qos::TopicQos & qos )  [inline]
```

Sets the TopicQos setting for this **Topic** (p. 2474).

Parameters

qos	Set of policies to be applied to the **Topic** (p. 2474). Immutable policies cannot be changed after this entity has been enabled.

Exceptions

dds::core::ImmutablePolicyError *(p. 1609)*	if `qos` contains a different value for an immutable policy.
dds::core::InconsistentPolicyError *(p. 1610)*	if some policies are inconsistent.

See also

dds::topic::qos::TopicQos (p. 2509) for rules on consistency among QoS policies

set_qos (abstract) (p. 1522)

dds::domain::DomainParticipant::default_topic_qos() (p. 1344)

dds::core::QosProvider::topic_qos() (p. 2032)

8.380.3.8 inconsistent_topic_status()

```
template<typename T >
dds::core::status::InconsistentTopicStatus  dds::topic::Topic< T >::inconsistent_topic_status ( )
const  [inline]
```

Retrieve the current InconsistentTopicStatus of this **Topic** (p. 2474).

8.380.4 Friends And Related Function Documentation

8.380.4.1 discover_any_topic() [1/2]

```
template<typename FwdIterator >
int32_t discover_any_topic (
            const  dds::domain::DomainParticipant & participant,
            FwdIterator begin,
            int32_t max_size )  [related]
```

Retrieves the topics discovered in a DomainParticipant.

Template Parameters

FwdIterator	A forward iterator whose value type is **dds::core::InstanceHandle** (p. 1612)

Parameters

participant	The DomainParticipant where to look up the discovered topics
begin	The beginning of the range where to insert the InstanceHandles that correspond to the discovered topics
max_size	The maximum number of topics to insert or **dds::core::LENGTH_UNLIMITED** (p. 438)

8.380.4.2 discover_any_topic() [2/2]

```
template<typename BinIterator >
int32_t discover_any_topic (
            const  dds::domain::DomainParticipant & participant,
            BinIterator begin )  [related]
```

Retrieves the topics discovered in a DomainParticipant.

This is equivalent to discover_any_topic(participant, begin, dds::core::LENGTH_UNLIMITED)

References **dds::topic::discover_any_topic()**, and **dds::core::LENGTH_UNLIMITED**.

8.380.4.3 discover_topic_data() [1/4]

```
template<typename T >
dds::topic::TopicBuiltinTopicData discover_topic_data (
            const  dds::domain::DomainParticipant & participant,
            const  dds::core::InstanceHandle & topic_handle )  [related]
```

Gets the information about a discovered topic.

Returns **dds::topic::TopicBuiltinTopicData** (p. 2493) for the specified **dds::topic::Topic** (p. 2474).

This operation retrieves information on a **dds::topic::Topic** (p. 2474) that has been discovered by the local Participant and must not have been "ignored" by means of the **dds::topic::ignore()** (p. 693) operation.

The `topic_handle` must correspond to such a topic. Otherwise, the operation will fail with **dds::core::PreconditionNotMetError** (p. 1941).

This call is not supported for remote topics. If a remote `topic_handle` is used, the operation will fail with **dds::core::UnsupportedError** (p. 2590).

Use the operation **dds::topic::discover_any_topic()** (p. 690) to find the topics that are currently discovered.

Parameters

participant	The DomainParticipant where to look up the topic information
topic_handle	<<***in***>> *(p. 355)* **dds::core::InstanceHandle** (p. 1612) of **dds::topic::Topic** (p. 2474).

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::PreconditionNotMetError** (p. 1941) or **dds::core::NotEnabledError** (p. 1869)

See also

dds::topic::TopicBuiltinTopicData (p. 2493)

dds::topic::discover_any_topic() (p. 690)

8.380.4.4 discover_topic_data() [2/4]

```
template<typename FwdIterator >
int32_t discover_topic_data (
            const  dds::domain::DomainParticipant & participant,
            FwdIterator begin,
            const  dds::core::InstanceHandleSeq & handles )  [related]
```

Gets the information about several topics.

8.380.4.5 discover_topic_data() [3/4]

```
template<typename FwdIterator >
int32_t discover_topic_data (
            const  dds::domain::DomainParticipant & participant,
            FwdIterator begin,
            int32_t max_size )  [related]
```

Gets the information about all topics.

Template Parameters

A	forward iterator whose value type is **dds::topic::TopicBuiltinTopicData** (p. 2493)

Parameters

participant	The DomainParticipant where to look up all the topics
begin	The beginning of a range where to copy the topic data
max_size	The maximum number of items to copy or **dds::core::LENGTH_UNLIMITED** (p. 438)

8.380.4.6 discover_topic_data() [4/4]

```
template<typename BinIterator >
int32_t discover_topic_data (
            const  dds::domain::DomainParticipant & participant,
            BinIterator begin )  [related]
```

Gets the information about all topics.

References **dds::topic::discover_topic_data()**, and **dds::core::LENGTH_UNLIMITED**.

8.380.4.7 ignore() [1/2]

```
template<typename T >
void ignore (
             dds::domain::DomainParticipant & participant,
            const  dds::core::InstanceHandle & handle )  [related]
```

Instructs a DomainParticipant to ignore a topic.

Instructs RTI Connext to locally ignore a **dds::topic::Topic** (p. 2474).

This means it will locally ignore any publication, or subscription to the **dds::topic::Topic** (p. 2474).

There is no way to reverse this operation.

This operation can be used to save local resources when the application knows that it will never publish or subscribe to data under certain topics.

The **dds::topic::Topic** (p. 2474) to ignore is identified by the `handle` argument. This is the handle of a **dds::topic::Topic** (p. 2474) that appears in the **dds::sub::SampleInfo** (p. 2285) retrieved when reading data samples from the built-in **dds::sub::DataReader** (p. 976) for the **dds::topic::Topic** (p. 2474).

Parameters

participant	The DomainParticipant where to ignore the topic
handle	<<***in***>> *(p. 355)* Handle of the **dds::topic::Topic** (p. 2474) to be ignored.

Exceptions

One	of the **Standard Exceptions** (p. 428), **dds::core::OutOfResourcesError** (p. 1898) or **dds::core::NotEnabledError** (p. 1869)

See also

dds::topic::TopicBuiltinTopicData (p. 2493)

dds::topic::topic_topic_name() (p. 442)

dds::sub::builtin_subscriber (p. 671)

8.380.4.8 ignore() [2/2]

```
template<typename FwdIterator >
void ignore (
              dds::domain::DomainParticipant & participant,
             FwdIterator begin,
             FwdIterator end )  [related]
```

Ignores a range of topics.

See also

ignore(dds::domain::DomainParticipant&,const dds::core::InstanceHandle&) (p. 2487)

References **dds::topic::ignore()**.

8.380.4.9 find()

```
template<typename TOPIC >
TOPIC find (
             const  dds::domain::DomainParticipant & participant,
             const std::string & topic_name )  [related]
```

Looks up a **Topic** (p. 2474) from a DomainParticipant.

Template Parameters

TOPIC	The topic to find; it can be one of the following: • A typed **TopicDescription** (p. 2503), such as `dds::topic::TopicDescription<Foo>`, • A typed **Topic** (p. 2474), such as `dds::topic::Topic<Foo>` • A typed **ContentFilteredTopic** (p. 954), such as `dds::topic::ContentFilteredTopic<Foo>` • **`dds::topic::AnyTopic`** (p. 825)

Parameters

participant	The DomainParticipant that contains the **Topic** (p. 2474)
topic_name	The topic name

Returns

A reference to an existing **Topic** (p. 2474) in this participant or an empty reference (i.e. equals to **dds::core::null** (p. 438)) if it doesn't exist.

Exceptions

dds::core::InvalidDowncastError *(p. 1620)*	If the concrete type of the topic description identified by `topic_name` is not `TOPIC`. For example, if the topic_name identifies a topic of type `Bar`, but the template parameter was dds::topic::Topic<Foo>.

The participant doesn't need to be enabled. The returned topic may be enabled or disabled.

8.380.4.10 find_topics() [1/2]

```
template<typename AnyTopicBackInsertIterator >
uint32_t find_topics (
             dds::domain::DomainParticipant participant,
            AnyTopicBackInsertIterator begin )  [related]
```

<<***extension***>> *(p. 354)* Retrieve all the **dds::topic::Topic** (p. 2474) created from this **dds::domain::Domain↩ Participant** (p. 1326)

```
#include <dds/topic/find.hpp>
```

Note

This is a standalone function in the namespace **rti::topic** (p. 771)

Template Parameters

AnyTopicBackInsertIterator	Type of the back-inserting iterator, whose value_type is **dds::topic::AnyTopic** (p. 825).

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Topics belong to
begin	A back-inserting iterator to the position in the destination container to insert the found Topics into

Returns

The number of found Topics

8.380.4.11 find_topics() [2/2]

```
template<typename AnyTopicForwardIterator >
uint32_t find_topics (
              dds::domain::DomainParticipant participant,
             AnyTopicForwardIterator begin,
             uint32_t max_size )  [related]
```

<<***extension***>> *(p. 354)* Retrieve all the **dds::topic::Topic** (p. 2474) created from this **dds::domain::Domain↩ Participant** (p. 1326)

```
#include <dds/topic/find.hpp>
```

Note

This is a standalone function in the namespace **rti::topic** (p. 771)

Template Parameters

AnyTopicBackInsertIterator	Type of the back-inserting iterator, whose value_type is **dds::topic::AnyTopic** (p. 825).

Parameters

participant	The **dds::domain::DomainParticipant** (p. 1326) the Topics belong to
begin	A forward iterator to the position in the destination container to insert the found Topics in
max_size	The maximum number of Topics to return

Returns

The number of found Topics

8.381 rti::topic::topic_type_disabled_copy< TopicType > Struct Template Reference

<<***extension***>> *(p. 354)* Indicates whether a TopicType is uncopyable

`#include <rti/topic/TopicTraits.hpp>`

Inherits dds::core::false_type.

8.381.1 Detailed Description

template<typename TopicType>
struct rti::topic::topic_type_disabled_copy< TopicType >

<<***extension***>> *(p. 354)* Indicates whether a TopicType is uncopyable

In general this is false_type (TopicType is a value types and therefore copyable), There is one situation when this is true_type:

- IDL types annotated with @language_binding(FLAT_DATA)

This trait enables (when it's `false_type`) or disables (when it's `true_type`) the DataReader read/take operations that copy instead of loaning the data (for example, **dds::sub::DataReader::take(SamplesFWIterator, int32_t)** (p. 1001))

8.382 rti::topic::topic_type_has_external_members< TopicType > Struct Template Reference

<<***extension***>> *(p. 354)* Indicates if a topic type contains directly or indirectly IDL external members.

`#include <rti/topic/TopicTraits.hpp>`

Inherits dds::core::false_type.

8.382.1 Detailed Description

template<typename TopicType>
struct rti::topic::topic_type_has_external_members< TopicType >

<<***extension***>> *(p. 354)* Indicates if a topic type contains directly or indirectly IDL external members.

This template is specialized in the generated code for topic types that do contain `&external` members.

8.383 dds::topic::topic_type_name< T > Struct Template Reference

Provides the name of a topic-type.

`#include <TopicTraits.hpp>`

8.383.1 Detailed Description

template<typename T>
struct dds::topic::topic_type_name< T >

Provides the name of a topic-type.

For example:

```
std::cout « dds::topic::topic_type_name<Foo>::value() « std::endl;
std::cout « dds::topic::topic_type_name<dds::core::StringTopicType>::value() « std::endl;
// output:
// Foo
// DDS::String
```

Note that `topic_type_name<dds::core::xtypes::DynamicData>::value()` is undefined, because DynamicData is used to dynamically represent any type so its actual topic-type is unknown at compilation time.

8.384 dds::topic::topic_type_support< T > Struct Template Reference

Provides convenience operations for a topic-type.

`#include <TopicTraits.hpp>`

8.384.1 Detailed Description

template<typename T>
struct dds::topic::topic_type_support< T >

Provides convenience operations for a topic-type.

8.385 dds::topic::TopicBuiltinTopicData Class Reference

Entry created when a **dds::topic::Topic** (p. 2474) object is discovered.

`#include <dds/topic/BuiltinTopic.hpp>`

Public Member Functions

- const **dds::topic::BuiltinTopicKey** & **key** () const

  Get the DCPS key to distinguish entries.
- const **dds::core::string** & **name** () const

  *Get the name of the **dds::topic::Topic** (p. 2474).*
- const **dds::core::string** & **type_name** () const

  *Get the name of the type attached to the **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::Durability** & **durability** () const

  *Get the **dds::core::policy::Durability** (p. 1433) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::DurabilityService** & **durability_service** () const

  *Get the **dds::core::policy::DurabilityService** (p. 1443) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::Deadline** & **deadline** () const

  *Get the **dds::core::policy::Deadline** (p. 1255) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::LatencyBudget** & **latency_budget** () const

  *Get the **dds::core::policy::LatencyBudget** (p. 1631) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::Liveliness** & **liveliness** () const

  *Get the **dds::core::policy::Liveliness** (p. 1645) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::Reliability** & **reliability** () const

  *Get the **dds::core::policy::Reliability** (p. 2149) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::TransportPriority** & **transport_priority** () const

  *Get the **dds::core::policy::TransportPriority** (p. 2552) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::Lifespan** & **lifespan** () const

  *Get the **dds::core::policy::Lifespan** (p. 1635) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::DestinationOrder** & **destination_order** () const

  *Get the **dds::core::policy::DestinationOrder** (p. 1258) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::History** & **history** () const

  *Get the **dds::core::policy::History** (p. 1602) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::ResourceLimits** & **resource_limits** () const

  *Get the **dds::core::policy::ResourceLimits** (p. 2203) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::Ownership** & **ownership** () const

  *Get the **dds::core::policy::Ownership** (p. 1899) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::TopicData** & **topic_data** () const

  *Get the **dds::core::policy::TopicData** (p. 2500) policy of the corresponding **dds::topic::Topic** (p. 2474).*
- const **dds::core::policy::DataRepresentation** & **representation** () const

  *Get the **dds::core::policy::DataRepresentation** (p. 1113) policy of the corresponding **dds::topic::Topic** (p. 2474).*

8.385.1 Detailed Description

Entry created when a **dds::topic::Topic** (p. 2474) object is discovered.

Data associated with the built-in topic **dds::topic::topic_topic_name()** (p. 442). It contains QoS policies and additional information that apply to the remote **dds::topic::Topic** (p. 2474).

Note: The **dds::topic::TopicBuiltinTopicData** (p. 2493) built-in topic is meant to convey information about discovered Topics. This **Topic** (p. 2474)'s samples are not propagated in a separate packet on the wire. Instead, the data is sent as part of the information carried by other built-in topics (**dds::topic::PublicationBuiltinTopicData** (p. 1977) and **dds↩ ::topic::SubscriptionBuiltinTopicData** (p. 2428)). Therefore **TopicBuiltinTopicData** (p. 2493) DataReaders will not receive any data.

8.385.2 Member Function Documentation

8.385.2.1 key()

```
const dds::topic::BuiltinTopicKey & dds::topic::TopicBuiltinTopicData::key ( ) const [inline]
```

Get the DCPS key to distinguish entries.

Returns

The key

References **dds::core::Value< D >::delegate()**.

8.385.2.2 name()

```
const dds::core::string & dds::topic::TopicBuiltinTopicData::name ( ) const [inline]
```

Get the name of the **dds::topic::Topic** (p. 2474).

Returns

The topic's name

References **dds::core::Value< D >::delegate()**.

8.385.2.3 type_name()

```
const dds::core::string & dds::topic::TopicBuiltinTopicData::type_name ( ) const [inline]
```

Get the name of the type attached to the **dds::topic::Topic** (p. 2474).

Returns

The type's name

References **dds::core::Value< D >::delegate()**.

8.385.2.4 durability()

```
const dds::core::policy::Durability & dds::topic::TopicBuiltinTopicData::durability ( ) const
[inline]
```

Get the **dds::core::policy::Durability** (p. 1433) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The Durability policy

8.385.2.5 durability_service()

```
const dds::core::policy::DurabilityService & dds::topic::TopicBuiltinTopicData::durability_↩
service ( ) const [inline]
```

Get the **dds::core::policy::DurabilityService** (p. 1443) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The DurabilityService policy

References **dds::core::Value**< **D** >**::delegate()**.

8.385.2.6 deadline()

```
const dds::core::policy::Deadline & dds::topic::TopicBuiltinTopicData::deadline ( ) const [inline]
```

Get the **dds::core::policy::Deadline** (p. 1255) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The Deadline policy

8.385.2.7 latency_budget()

```
const dds::core::policy::LatencyBudget & dds::topic::TopicBuiltinTopicData::latency_budget ( )
const [inline]
```

Get the **dds::core::policy::LatencyBudget** (p. 1631) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The LatencyBudget policy

References **dds::core::Value< D >::delegate()**.

8.385.2.8 liveliness()

```
const dds::core::policy::Liveliness & dds::topic::TopicBuiltinTopicData::liveliness ( ) const
[inline]
```

Get the **dds::core::policy::Liveliness** (p. 1645) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The Liveliness policy

References **dds::core::Value< D >::delegate()**.

8.385.2.9 reliability()

```
const dds::core::policy::Reliability & dds::topic::TopicBuiltinTopicData::reliability ( ) const
[inline]
```

Get the **dds::core::policy::Reliability** (p. 2149) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The Reliability policy

References **dds::core::Value< D >::delegate()**.

8.385.2.10 transport_priority()

```
const dds::core::policy::TransportPriority & dds::topic::TopicBuiltinTopicData::transport_↩
priority ( ) const [inline]
```

Get the **dds::core::policy::TransportPriority** (p. 2552) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The TransportPriority policy

References **dds::core::Value**< **D** >**::delegate()**.

8.385.2.11 lifespan()

```
const dds::core::policy::Lifespan & dds::topic::TopicBuiltinTopicData::lifespan ( ) const [inline]
```

Get the **dds::core::policy::Lifespan** (p. 1635) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The Lifespan policy

References **dds::core::Value**< **D** >**::delegate()**.

8.385.2.12 destination_order()

```
const dds::core::policy::DestinationOrder & dds::topic::TopicBuiltinTopicData::destination_order
( ) const [inline]
```

Get the **dds::core::policy::DestinationOrder** (p. 1258) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The DestinationOrder policy

References **dds::core::Value**< **D** >**::delegate()**.

8.385.2.13 history()

```
const dds::core::policy::History & dds::topic::TopicBuiltinTopicData::history ( ) const [inline]
```

Get the **dds::core::policy::History** (p. 1602) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The History policy

References **dds::core::Value**< **D** >**::delegate()**.

8.385.2.14 resource_limits()

```
const dds::core::policy::ResourceLimits & dds::topic::TopicBuiltinTopicData::resource_limits ( )
const [inline]
```

Get the **dds::core::policy::ResourceLimits** (p. 2203) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The ResourceLimits policy

References **dds::core::Value**< **D** >**::delegate()**.

8.385.2.15 ownership()

```
const dds::core::policy::Ownership & dds::topic::TopicBuiltinTopicData::ownership ( ) const [inline]
```

Get the **dds::core::policy::Ownership** (p. 1899) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The Ownership policy

References **dds::core::Value**< **D** >**::delegate()**.

8.385.2.16 topic_data()

```
const dds::core::policy::TopicData & dds::topic::TopicBuiltinTopicData::topic_data ( ) const
[inline]
```

Get the **dds::core::policy::TopicData** (p. 2500) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The TopicData policy

References **dds::core::Value< D >::delegate()**.

8.385.2.17 representation()

```
const dds::core::policy::DataRepresentation & dds::topic::TopicBuiltinTopicData::representation (
) const [inline]
```

Get the **dds::core::policy::DataRepresentation** (p. 1113) policy of the corresponding **dds::topic::Topic** (p. 2474).

Returns

The DataRepresentation policy

References **dds::core::Value< D >::delegate()**.

8.386 dds::core::policy::TopicData Class Reference

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **TopicData** ()
  Creates an empty sequence of bytes.
- **TopicData** (const **dds::core::ByteSeq** &seq)
  Creates an instance from a vector of bytes.
- **TopicData** (const uint8_t *value_begin, const uint8_t *value_end)
  Creates an instance from a range of bytes.
- template<typename OCTET_ITER >
  TopicData & **value** (OCTET_ITER the_begin, OCTET_ITER the_end)
  Set the value for the topic data.
- const **dds::core::ByteSeq** **value** () const
  Get the topic data.
- **dds::core::ByteSeq** & **value** (**dds::core::ByteSeq** &dst) const
  Get the topic data.
- const uint8_t * **begin** () const
  Beginning of the range of bytes.
- const uint8_t * **end** () const
  End of the range of bytes.

8.386.1 Detailed Description

Entity:

dds::topic::Topic (p. 2474)

Properties:

RxO (p. 514) = NO
Changeable (p. 514) = **YES** (p. 514)

See also

dds::sub::builtin_subscriber (p. 671)

8.386.2 Usage

The purpose of this QoS is to allow the application to attach additional information to the created **dds::topic::Topic** (p. 2474) objects, so that when a remote application discovers their existence, it can access that information and use it for its own purposes. This extra data is not used by RTI Connext.

One possible use of this QoS is to attach security credentials or some other information that can be used by the remote application to authenticate the source.

In combination with **dds::sub::DataReaderListener** (p. 1055), **dds::pub::DataWriterListener** (p. 1202), or operations such as **dds::topic::ignore()** (p. 693), this QoS policy can assist an application in defining and enforcing its own security policies.

The use of this QoS is not limited to security; it offers a simple, yet flexible extensibility mechanism.

Important: RTI Connext stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Connext with the maximum size of the data that will be stored in policies of this type. This size is configured with **rti::core::policy::DomainParticipantResourceLimits::topic_data_max_length** (p. 1414).

See also

UserData (p. 2591)

8.386.3 Constructor & Destructor Documentation

8.386.3.1 TopicData() [1/3]

```
dds::core::policy::TopicData::TopicData ( )  [inline]
```

Creates an empty sequence of bytes.

8.386.3.2 TopicData() [2/3]

```
dds::core::policy::TopicData::TopicData (
            const  dds::core::ByteSeq & seq )  [inline], [explicit]
```

Creates an instance from a vector of bytes.

8.386.3.3 TopicData() [3/3]

```
dds::core::policy::TopicData::TopicData (
            const uint8_t * value_begin,
            const uint8_t * value_end )  [inline]
```

Creates an instance from a range of bytes.

8.386.4 Member Function Documentation

8.386.4.1 value() [1/3]

```
template<typename OCTET_ITER >
TopicData & dds::core::policy::TopicData::value (
            OCTET_ITER the_begin,
            OCTET_ITER the_end )  [inline]
```

Set the value for the topic data.

8.386.4.2 value() [2/3]

```
const  dds::core::ByteSeq dds::core::policy::TopicData::value ( ) const  [inline]
```

Get the topic data.

8.386.4.3 value() [3/3]

```
 dds::core::ByteSeq & dds::core::policy::TopicData::value (
             dds::core::ByteSeq & dst ) const  [inline]
```

Get the topic data.

8.386.4.4 begin()

```
const uint8_t * dds::core::policy::TopicData::begin ( ) const  [inline]
```

Beginning of the range of bytes.

8.386.4.5 end()

```
const uint8_t * dds::core::policy::TopicData::end ( ) const  [inline]
```

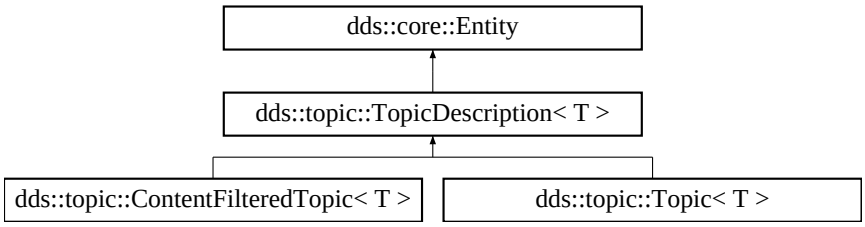
End of the range of bytes.

8.387 dds::topic::TopicDescription< T > Class Template Reference

Abstract base class of **Topic** (p. 2474) and **ContentFilteredTopic** (p. 954).

```
#include <dds/topic/TopicDescription.hpp>
```

Inheritance diagram for dds::topic::TopicDescription< T >:



Public Member Functions

- const std::string & **name** () const

  Gets the topic name.
- const std::string & **type_name** () const

  Gets the type name.
- const **dds::domain::DomainParticipant** & **participant** () const

  *Gets the related **dds::domain::DomainParticipant** (p. 1326).*

Additional Inherited Members

8.387.1 Detailed Description

template<typename T>
class dds::topic::TopicDescription< T >

Abstract base class of **Topic** (p. 2474) and **ContentFilteredTopic** (p. 954).

TopicDescription (p. 2503) represents the fact that both publications and subscriptions are tied to a single data-type. Its attribute `type_name` defines a unique resulting type for the publication or the subscription and therefore creates an implicit association with a TypeSupport.

TopicDescription (p. 2503) has also a `name` that allows it to be retrieved locally.

See also

TypeSupport, FooTypeSupport

Template Parameters

T	The topic-type

8.387.2 Member Function Documentation

8.387.2.1 name()

```
template<typename T >
const std::string &  dds::topic::TopicDescription< T >::name ( ) const  [inline]
```

Gets the topic name.

8.387.2.2 type_name()

```
template<typename T >
const std::string &  dds::topic::TopicDescription< T >::type_name ( ) const  [inline]
```

Gets the type name.

The type name defines a locally unique type for the publication or the subscription.

The association between the topic and the type name is automatic through the template parameter `T`, via the trait dds::topic::topic_type_name<T>::name(), that is given

When `T` is rti::core::xtypes::DynamicData, the type name is obtained by default from the rti::core::xtypes::DynamicType.

8.387.2.3 participant()

```
template<typename T >
const dds::domain::DomainParticipant & dds::topic::TopicDescription< T >::participant ( ) const
[inline]
```

Gets the related **dds::domain::DomainParticipant** (p. 1326).

8.388 dds::topic::TopicInstance< T > Class Template Reference

Encapsulates a sample and its associated instance handle.

```
#include <TopicInstance.hpp>
```

Public Member Functions

- **TopicInstance** ()
  *Create a null **TopicInstance** (p. 2505).*
- **TopicInstance** (const **::dds::core::InstanceHandle** &the_handle)
  *Create a **TopicInstance** (p. 2505) with an InstanceHandle.*
- **TopicInstance** (const **::dds::core::InstanceHandle** &the_handle, const T &the_sample)
  *Create a **TopicInstance** (p. 2505) with a data sample and an InstanceHandle.*
- **operator dds::core::InstanceHandle** () const
  Obtains the InstanceHandle.
- **dds::core::InstanceHandle handle** () const
  Obtains the InstahnceHandle.
- void **handle** (const **::dds::core::InstanceHandle** &the_handle)
  Sets the InstanceHandle.
- const T & **sample** () const
  Gets the sample by const reference.
- T & **sample** ()
  Gets the sample by reference.
- void **sample** (const T &the_sample)
  Assigns the data sample.

8.388.1 Detailed Description

template<typename T>
class dds::topic::TopicInstance< T >

Encapsulates a sample and its associated instance handle.

Template Parameters

The	type of the sample that the **TopicInstance** (p. 2505) is encapsulating.

See also

dds::pub::DataWriter::write(const dds::topic::TopicInstance<T>&) (p. 1154)

8.388.2 Constructor & Destructor Documentation

8.388.2.1 TopicInstance() [1/3]

```
template<typename T >
dds::topic::TopicInstance< T >::TopicInstance ( )  [inline]
```

Create a null **TopicInstance** (p. 2505).

The sample is default constructed and the instance handle is equal to **dds::core::null** (p. 438).

8.388.2.2 TopicInstance() [2/3]

```
template<typename T >
dds::topic::TopicInstance< T >::TopicInstance (
            const  ::dds::core::InstanceHandle & the_handle )  [inline]
```

Create a **TopicInstance** (p. 2505) with an InstanceHandle.

The sample is default constructed and the instance handle is copied from `handle`.

8.388.2.3 TopicInstance() [3/3]

```
template<typename T >
dds::topic::TopicInstance< T >::TopicInstance (
            const  ::dds::core::InstanceHandle & the_handle,
            const T & the_sample )  [inline]
```

Create a **TopicInstance** (p. 2505) with a data sample and an InstanceHandle.

8.388.3 Member Function Documentation

8.388.3.1 operator dds::core::InstanceHandle()

```
template<typename T >
dds::topic::TopicInstance< T >::operator  dds::core::InstanceHandle ( ) const  [inline]
```

Obtains the InstanceHandle.

References **dds::topic::TopicInstance< T >::handle()**.

8.388.3.2 handle() [1/2]

```
template<typename T >
dds::core::InstanceHandle  dds::topic::TopicInstance< T >::handle ( ) const  [inline]
```

Obtains the InstahnceHandle.

Referenced by **dds::topic::TopicInstance< T >::operator dds::core::InstanceHandle()**.

8.388.3.3 handle() [2/2]

```
template<typename T >
void  dds::topic::TopicInstance< T >::handle (
            const  ::dds::core::InstanceHandle & the_handle )  [inline]
```

Sets the InstanceHandle.

8.388.3.4 sample() [1/3]

```
template<typename T >
const T &  dds::topic::TopicInstance< T >::sample ( ) const  [inline]
```

Gets the sample by const reference.

8.388.3.5 sample() [2/3]

```
template<typename T >
T &  dds::topic::TopicInstance< T >::sample ( )  [inline]
```

Gets the sample by reference.

8.388.3.6 sample() [3/3]

```
template<typename T >
void  dds::topic::TopicInstance< T >::sample (
            const T & the_sample )  [inline]
```

Assigns the data sample.

Parameters

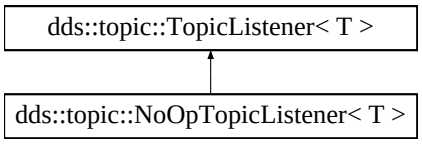
the_sample	The data sample to copy into this **TopicInstance** (p. 2505)

8.389 dds::topic::TopicListener< T > Class Template Reference

The listener to notify status changes for a **dds::topic::Topic** (p. 2474).

`#include <dds/topic/TopicListener.hpp>`

Inheritance diagram for dds::topic::TopicListener< T >:



Public Member Functions

- virtual void **on_inconsistent_topic** (**Topic**< T > &topic, const **dds::core::status::InconsistentTopicStatus** &status)=0

  *Handles the **dds::core::status::InconsistentTopicStatus** (p. 1611) status.*

8.389.1 Detailed Description

template<typename T>
class dds::topic::TopicListener< T >

The listener to notify status changes for a **dds::topic::Topic** (p. 2474).

Entity:

dds::topic::Topic (p. 2474)

Status:

dds::core::status::StatusMask::inconsistent_topic() (p. 2379), **dds::core::status::InconsistentTopicStatus** (p. 1611)

This is the interface that can be implemented by an application-provided class and then registered with the **dds::topic::Topic** (p. 2474) such that the application can be notified by RTI Connext of relevant status changes.

See also

Status Kinds (p. 429)

Listener (p. 1637)

dds::topic::Topic::set_listener (p. 2482)

Operations Allowed in Listener Callbacks (p. 1639)

NoOpTopicListener (p. 1867)

8.389.2 Member Function Documentation

8.389.2.1 on_inconsistent_topic()

```
template<typename T >
virtual void  dds::topic::TopicListener< T >::on_inconsistent_topic (
              Topic< T > & topic,
             const  dds::core::status::InconsistentTopicStatus & status )  [pure virtual]
```

Handles the **dds::core::status::InconsistentTopicStatus** (p. 1611) status.

This callback is called when a remote **dds::topic::Topic** (p. 2474) is discovered but is inconsistent with the locally created **dds::topic::Topic** (p. 2474) of the same topic name.

Parameters

topic	<<***out***>> *(p. 355)* Locally created **dds::topic::Topic** (p. 2474) that triggers the listener callback
status	<<***out***>> *(p. 355)* Current inconsistent status of locally created **dds::topic::Topic** (p. 2474)

Implemented in **dds::topic::NoOpTopicListener**< **T** > (p. 1867).

8.390 dds::topic::qos::TopicQos Class Reference

<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::topic::Topic** (p. 2474) supports

`#include <dds/topic/qos/TopicQos.hpp>`

Public Member Functions

- **TopicQos** ()
  *Creates a **TopicQos** (p. 2509) with the default value for each policy.*
- template<typename POLICY >
  const POLICY & **policy** () const
  Gets a QoS policy by const reference.
- template<typename POLICY >
  POLICY & **policy** ()
  Gets a QoS policy by reference.
- template<typename Policy >
  TopicQos & **policy** (const Policy &p)
  Sets a policy.
- template<typename Policy >
  TopicQos & **operator**<< (const Policy &p)
  Sets a policy.
- template<typename Policy >
  const **TopicQos** & **operator**>> (Policy &p) const
  Copies the values of a policy.

Related Functions

(Note that these are not member functions.)

- std::string **to_string** (const **TopicQos** &qos, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrint↩ Format**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::topic::qos::TopicQos*** *(p. 2509)*
- std::string **to_string** (const **TopicQos** &qos, const **TopicQos** &base, const **rti::core::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::topic::qos::TopicQos*** *(p. 2509)*
- std::string **to_string** (const **TopicQos** &qos, const **rti::core::qos_print_all_t** &qos_print_all, const **rti::core↩ ::QosPrintFormat** &format= **rti::core::QosPrintFormat**())

  <<***extension***>> *(p. 354) Obtains a string representation of the* ***dds::topic::qos::TopicQos*** *(p. 2509)*
- std::ostream & **operator**<< (std::ostream &out, const rti::topic::qos::TopicQos &qos)

  <<***extension***>> *(p. 354) Prints a* ***dds::topic::qos::TopicQos*** *(p. 2509) to an output stream.*

8.390.1 Detailed Description

<<***value-type***>> *(p. 350)* Container of the QoS policies that a **dds::topic::Topic** (p. 2474) supports

8.390.2 TopicQos Policies

A **TopicQos** (p. 2509) contains the following policies:

- **dds::core::policy::TopicData** (p. 2500),
- **dds::core::policy::Durability** (p. 1433),
- **dds::core::policy::DurabilityService** (p. 1443),
- **dds::core::policy::Deadline** (p. 1255),
- **dds::core::policy::LatencyBudget** (p. 1631),
- **dds::core::policy::Liveliness** (p. 1645),
- **dds::core::policy::Reliability** (p. 2149),
- **dds::core::policy::DestinationOrder** (p. 1258),
- **dds::core::policy::History** (p. 1602),
- **dds::core::policy::ResourceLimits** (p. 2203),
- **dds::core::policy::TransportPriority** (p. 2552),
- **dds::core::policy::Lifespan** (p. 1635),
- **dds::core::policy::Ownership** (p. 1899),
- **dds::core::policy::DataRepresentation** (p. 1113)

To get or set policies use the **policy()** (p. 2512) getters and setters or operator << (see **examples** (p. 603)).

You must set certain members in a consistent manner:

length of dds::topic::qos::TopicQos::topic_data .value <= dds::domain::qos::DomainParticipantQos::resource_limits .topic_data_max_length

If any of the above are not true, **dds::topic::Topic::qos(const dds::topic::qos::TopicQos&)** (p. 2484), **dds::topic::Topic::qos()** (p. 2483) and **dds::domain::DomainParticipant::default_topic_qos** (p. 1344) will fail with **dds::core::InconsistentPolicyError** (p. 1610) and the **dds::topic::Topic** (p. 2474) constructors will fail with **dds::core::Error** (p. 1539)

Entity:

dds::topic::Topic (p. 2474)

See also

All QoS Policies (p. 508) allowed ranges within each Qos.

Qos Use Cases (p. 602)

8.390.3 Constructor & Destructor Documentation

8.390.3.1 TopicQos()

```
dds::topic::qos::TopicQos::TopicQos ( )  [inline]
```

Creates a **TopicQos** (p. 2509) with the default value for each policy.

8.390.4 Member Function Documentation

8.390.4.1 policy() [1/3]

```
template<typename POLICY >
const POLICY & dds::topic::qos::TopicQos::policy ( ) const
```

Gets a QoS policy by const reference.

Template Parameters

Policy	One of the **TopicQos Policies** (p. 2510)

See also

Setting Qos Values (p. 603)

8.390.4.2 policy() [2/3]

```
template<typename POLICY >
POLICY & dds::topic::qos::TopicQos::policy ( )
```

Gets a QoS policy by reference.

Template Parameters

Policy	One of the **TopicQos Policies** (p. 2510)

See also

Setting Qos Values (p. 603)

8.390.4.3 policy() [3/3]

```
template<typename Policy >
TopicQos & dds::topic::qos::TopicQos::policy (
            const Policy & p )  [inline]
```

Sets a policy.

See also

policy() (p. 2512)

Setting Qos Values (p. 603)

8.390.4.4 operator<<()

```
template<typename Policy >
TopicQos & dds::topic::qos::TopicQos::operator<< (
            const Policy & p )  [inline]
```

Sets a policy.

See also

policy() (p. 2512)

Setting Qos Values (p. 603)

8.390.4.5 operator>>()

```
template<typename Policy >
const  TopicQos & dds::topic::qos::TopicQos::operator>> (
            Policy & p ) const  [inline]
```

Copies the values of a policy.

Parameters

p	The destination where to copy the current value of the Policy

See also

policy() (p. 2512)

Setting Qos Values (p. 603)

References **rti::topic::to_string()**.

8.390.5 Friends And Related Function Documentation

8.390.5.1 to_string() [1/3]

```
std::string to_string (
            const  TopicQos & qos,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::topic::qos::TopicQos** (p. 2509)

Parameters

qos	The qos object to convert to a string
format	The optional print format used to format the output string.

The several to_string overloads allow formatting the output and printing only the differences with respect to another Qos object.

```
TopicQos qos;
// When no QosPrintFormat is supplied, the default is used. Similarly, since
// no base profile has been specified we will only print the differences
// with respect to the documented default for TopicQos
std::string str = to_string(qos);
// In this overload we are specifying the print format used to format the output
// string.
QosPrintFormat format; // ...;
str = to_string(qos, format);
// Here, the differences between qos and base will be included in the output
// string. If the two qos objects are the same, the resulting string will
// be empty.
TopicQos base; // ...;
str = to_string(qos, base);
// We could also specify the format at this point
str = to_string(qos, base, format);
// Instead of supplying a base profile, the sentinel value qos_print_all can
// be supplied. This will result in the entire Qos object being printed (as
// opposed to only the differences with respect to a base qos).
str = to_string(qos, qos_print_all);
```

This overload uses the default print format and only prints the differences between qos and the documented default.

Returns

The string representation of the qos

8.390.5.2 to_string() [2/3]

```
std::string to_string (
            const  TopicQos & qos,
            const  TopicQos & base,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::topic::qos::TopicQos** (p. 2509)

Parameters

qos	The qos object to convert to a string
base	The base qos. Only the differences between base and qos are included in the output string. If you want to print everything within the QoS, use the overload that accepts a **rti::core::qos_print_all_t** (p. 2019).
format	The optional print format used to format the output string.

This overload prints the differences between qos and base. If no print format is specified, the default will be used.

Returns

The string representation of the qos

8.390.5.3 to_string() [3/3]

```
std::string to_string (
            const  TopicQos & qos,
            const  rti::core::qos_print_all_t & qos_print_all,
            const  rti::core::QosPrintFormat & format =  rti::core::QosPrintFormat() )  [related]
```

<<***extension***>> *(p. 354)* Obtains a string representation of the **dds::topic::qos::TopicQos** (p. 2509)

Parameters

qos	The qos object to convert to a string
qos_print_all	The sentinel value indicating that the entire QoS should be converted to a string. The other overloads of this API only print the differences with respect to a base profile, or to the documented default.
format	The optional print format used to format the output string.

This overload prints all of the policies within qos. If no print format is specified, the default will be used.

The only valid value for the qos_print_all argument is **rti::core::qos_print_all** (p. 439)

For example:
```
to_string(qos, rti::core::qos_print_all);
```

Returns

The string representation of the qos

8.390.5.4 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const rti::topic::qos::TopicQos & qos )  [related]
```

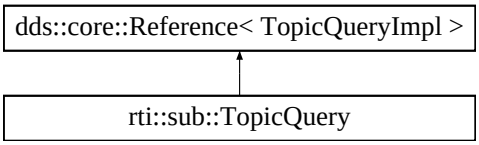
<<***extension***>> *(p. 354)* Prints a **dds::topic::qos::TopicQos** (p. 2509) to an output stream.

8.391 rti::sub::TopicQuery Class Reference

*<<**extension**>> (p. 354) <<**reference-type**>> (p. 351)* Allows a **dds::sub::DataReader** (p. 976) to query the sample cache of its matching dds::pub::DataWriters.

`#include <rti/sub/TopicQuery.hpp>`

Inheritance diagram for rti::sub::TopicQuery:



Public Member Functions

- **TopicQuery** (**dds::sub::AnyDataReader** reader, const **TopicQuerySelection** &selection)

  *Creates a **TopicQuery** (p. 2516) for a given DataReader.*
- void **close** ()

  *Deletes and cancels this **TopicQuery** (p. 2516).*
- const **rti::core::Guid guid** () const

  *Gets the **TopicQuery** (p. 2516) GUID.*
- bool **closed** () const

  *Indicates whether this **TopicQuery** (p. 2516) has been closed with **close()** (p. 2518).*
- **dds::sub::AnyDataReader datareader** () const

  *Gets the DataReader associated to this **TopicQuery** (p. 2516).*

Static Public Member Functions

- static **TopicQuery UseReaderContentFilter** (**dds::sub::AnyDataReader** reader)

  *Creates a **TopicQuery** (p. 2516) with the same filter as this reader's ContentFilteredTopic.*
- static **TopicQuery SelectAll** (**dds::sub::AnyDataReader** reader)

  *Creates a **TopicQuery** (p. 2516) that requests all the samples in the DataWriters' cache.*

Related Functions

(Note that these are not member functions.)

- **TopicQuery find_topic_query** (**dds::sub::AnyDataReader datareader**, const **rti::core::Guid** & **guid**)

  *Looks up a **TopicQuery** (p. 2516) by its GUID.*

8.391.1 Detailed Description

<<***extension***>> *(p. 354)* <<***reference-type***>> *(p. 351)* Allows a **dds::sub::DataReader** (p. 976) to query the sample cache of its matching dds::pub::DataWriters.

Note

A **TopicQuery** (p. 2516) provides all the functions of a <<***reference-type***>> *(p. 351)*, including **close()** (p. 2518) and retain().

You can create a **TopicQuery** (p. 2516) for a DataReader with the constructor that receives a **TopicQuerySelection** (p. 2526) or with the following static member functions:

- **TopicQuery::UseReaderContentFilter** (p. 2518)
- **TopicQuery::SelectAll** (p. 2519)

As a <<***reference-type***>> *(p. 351)*, a **TopicQuery** (p. 2516) will be deleted when all references to it go out of scope or after calling **close()** (p. 2518). The middleware will notify the deletion of a **TopicQuery** (p. 2516) to the DataWriters. See **close()** (p. 2518) for more details.

8.391.2 Constructor & Destructor Documentation

8.391.2.1 TopicQuery()

```
rti::sub::TopicQuery::TopicQuery (
              dds::sub::AnyDataReader reader,
             const  TopicQuerySelection & selection )  [inline]
```

Creates a **TopicQuery** (p. 2516) for a given DataReader.

The resulting **TopicQuery** (p. 2516) will propagate to the matching DataWriters right after creating if the reader is enabled. Otherwise it will propagate after enabling the reader.

Any late-joining DataWriter matching with the DataReader will also receive the query.

Parameters

reader	The DataReader that will query and receive the requested samples
selection	Selects which samples the matching DataWriters will provide.

There are two special kinds of TopicQueries, created with the static member functions **UseReaderContentFilter()** (p. 2518) and **SelectAll()** (p. 2519).

8.391.3 Member Function Documentation

8.391.3.1 close()

```
void rti::sub::TopicQuery::close ( )  [inline]
```

Deletes and cancels this **TopicQuery** (p. 2516).

This is a explicit deletion, which otherwise would occur when all references to this **TopicQuery** (p. 2516) go out of scope.

After deleting a **TopicQuery** (p. 2516) new DataWriters won't discover it and existing DataWriters currently publishing cached samples may stop before delivering all of them.

8.391.3.2 guid()

```
const  rti::core::Guid rti::sub::TopicQuery::guid ( ) const  [inline]
```

Gets the **TopicQuery** (p. 2516) GUID.

See also

rti::sub::find_topic_query() (p. 769)

8.391.3.3 closed()

```
bool rti::sub::TopicQuery::closed ( ) const  [inline]
```

Indicates whether this **TopicQuery** (p. 2516) has been closed with **close()** (p. 2518).

8.391.3.4 datareader()

```
dds::sub::AnyDataReader rti::sub::TopicQuery::datareader ( ) const  [inline]
```

Gets the DataReader associated to this **TopicQuery** (p. 2516).

8.391.3.5 UseReaderContentFilter()

```
static  TopicQuery rti::sub::TopicQuery::UseReaderContentFilter (
              dds::sub::AnyDataReader reader )  [static]
```

Creates a **TopicQuery** (p. 2516) with the same filter as this reader's ContentFilteredTopic.

If the reader doesn't use a **dds::topic::ContentFilteredTopic** (p. 954), this **TopicQuery** (p. 2516) behaves as **TopicQuery::SelectAll()** (p. 2519).

8.391.3.6 SelectAll()

```
static  TopicQuery rti::sub::TopicQuery::SelectAll (
              dds::sub::AnyDataReader reader )  [static]
```

Creates a **TopicQuery** (p. 2516) that requests all the samples in the DataWriters' cache.

8.391.4 Friends And Related Function Documentation

8.391.4.1 find_topic_query()

```
TopicQuery find_topic_query (
              dds::sub::AnyDataReader datareader,
             const  rti::core::Guid & guid )  [related]
```

Looks up a **TopicQuery** (p. 2516) by its GUID.

Parameters

datareader	The DataReader used to create the **TopicQuery** (p. 2516)
guid	The **TopicQuery** (p. 2516)'s GUID

Returns

The **TopicQuery** (p. 2516) if it exists or a **dds::core::null** (p. 438) reference otherwise.

See also

rti::sub::TopicQuery::guid() (p. 2518)

8.392 rti::sub::TopicQueryData Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Provides information about a **TopicQuery** (p. 2516)

`#include <rti/sub/TopicQuery.hpp>`

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **TopicQuerySelection selection** () const
  The data selection.
- const **dds::core::string** & **topic_name** () const
  *The topic name of the **dds::sub::DataReader** (p. 976).*
- const **rti::core::Guid** & **original_related_reader_guid** () const
  *Identifies the **dds::sub::DataReader** (p. 976) that created the **rti::sub::TopicQuery** (p. 2516).*

Related Functions

(Note that these are not member functions.)

- **TopicQueryData create_topic_query_data_from_service_request** (const **rti::topic::ServiceRequest** &service_request)
  *Creates a **TopicQueryData** (p. 2520) from a ServiceRequest.*

8.392.1 Detailed Description

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Provides information about a **TopicQuery** (p. 2516)

Contains the information about a **TopicQuery** (p. 2516) that can be retrieved using **rti::sub::create_topic_query_↩ data_from_service_request()** (p. 768).

See also

rti::sub::create_topic_query_data_from_service_request() (p. 768)

8.392.2 Member Function Documentation

8.392.2.1 selection()

```
TopicQuerySelection rti::sub::TopicQueryData::selection ( ) const
```

The data selection.

8.392.2.2 topic_name()

```
const  dds::core::string & rti::sub::TopicQueryData::topic_name ( ) const
```

The topic name of the **dds::sub::DataReader** (p. 976).

8.392.2.3 original_related_reader_guid()

```
const  rti::core::Guid & rti::sub::TopicQueryData::original_related_reader_guid ( ) const
```

Identifies the **dds::sub::DataReader** (p. 976) that created the **rti::sub::TopicQuery** (p. 2516).

8.392.3 Friends And Related Function Documentation

8.392.3.1 create_topic_query_data_from_service_request()

```
TopicQueryData create_topic_query_data_from_service_request (
            const  rti::topic::ServiceRequest & service_request )  [related]
```

Creates a **TopicQueryData** (p. 2520) from a ServiceRequest.

This operation will extract the content from the request body of the **rti::topic::ServiceRequest** (p. 2358) to create a **rti::sub::TopicQueryData** (p. 2520) object.

The specified **rti::topic::ServiceRequest** (p. 2358) must be a valid sample associated with the service id **rti::core::ServiceRequestId_def::TOPIC_QUERY** (p. 2362). Otherwise this operation will return false.

This operation can be called within the context of a **dds::pub::DataWriterListener::on_service_request_accepted** (p. 1209) to retrieve the **rti::sub::TopicQueryData** (p. 2520) of a **rti::topic::ServiceRequest** (p. 2358) that has been received with service id **rti::core::ServiceRequestId_def::TOPIC_QUERY** (p. 2362)

Parameters

service_request	The **rti::topic::ServiceRequest** (p. 2358) that contains the **rti::sub::TopicQueryData** (p. 2520) as part of its request body.

Returns

A **rti::sub::TopicQueryData** (p. 2520) object where the content from the service request is extracted.

8.393 rti::core::policy::TopicQueryDispatch Class Reference

<<***extension***>> *(p. 116)* Configures the ability of a **dds::pub::DataWriter** (p. 1138) to publish samples in response to a **rti::sub::TopicQuery** (p. 2516)

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **TopicQueryDispatch** ()

  Creates the default policy.
- **TopicQueryDispatch** (bool the_enable, const **dds::core::Duration** &the_publication_period, int32_t the_↩ samples_per_period)

  Creates a policy with the provided values for enable, publication_period and samples_per_period.
- **TopicQueryDispatch** & **enable** (bool the_enable)

  <<***extension***>> *(p. 116) Allows this writer to dispatch TopicQueries*
- bool **enable** () const

  Getter (see setter with the same name)
- **TopicQueryDispatch** & **publication_period** (const **dds::core::Duration** &the_publication_period)

  <<***basic***>> *(p. 104)* <<***extension***>> *(p. 116) Sets the periodic interval at which samples are published.*
- **dds::core::Duration** **publication_period** () const

  Getter (see setter with the same name)
- **TopicQueryDispatch** & **samples_per_period** (int32_t the_samples_per_period)

  <<***extension***>> *(p. 116) Sets the maximum number of samples to publish in each publication_period.*
- int32_t **samples_per_period** () const

  Getter (see setter with the same name)

8.393.1 Detailed Description

<<***extension***>> *(p. 116)* Configures the ability of a **dds::pub::DataWriter** (p. 1138) to publish samples in response to a **rti::sub::TopicQuery** (p. 2516)

Enables the ability of a **dds::pub::DataWriter** (p. 1138) to publish historical samples upon reception of a **rti::sub::TopicQuery** (p. 2516) and how often they are published.

Since a TopicQuery selects previously written samples, the DataWriter must have a **dds::core::policy::Durability::kind** (p. 1437) different from **dds::core::policy::DurabilityKind_def::VOLATILE** (p. 1442). Also, **dds::core::policy::Reliability::kind** (p. 2153) must be set to **dds::core::policy::ReliabilityKind_def::RELIABLE** (p. 2157).

Only samples that fall within the **dds::core::policy::Durability::writer_depth** (p. 1439) for an instance are evaluated against the TopicQuery filter. While the DataWriter is waiting for acknowledgements from one or more DataReaders, there may temporarily be more than **dds::core::policy::Durability::writer_depth** (p. 1439) samples per instance in the DataWriter's queue if the **dds::core::policy::History::depth** (p. 1605) is set to a higher value than **dds::core::policy::Durability::writer_depth** (p. 1439). Those additional samples past **dds::core::policy::Durability::writer_depth** (p. 1439) are not eligible to be sent in response to the TopicQuery.

A TopicQuery may select multiple samples at once. The writer will publish them periodically, independently from newly written samples. **rti::core::policy::TopicQueryDispatch::publication_period** (p. 2525) configures the frequency of that period and **rti::core::policy::TopicQueryDispatch::samples_per_period** (p. 2525) configures the maximum number of samples to publish each period.

If the DataWriter blocks during the publication of one of these samples, it will stop and try again the next period. (See **dds::pub::DataWriter::write()** (p. 1148) for the conditions that may cause the write operation to block.)

All the DataWriters that belong to a single **dds::pub::Publisher** (p. 1993) and enable TopicQueries share the same event thread, but each DataWriter schedules separate events. To configure that thread, see **rti::core::policy::AsynchronousPublisher::topic_query_publication_thread** (p. 838).

If the DataWriter is dispatching more than one TopicQuery at the same time, the configuration of this periodic event applies to all of them. For example, if a DataWriter receives two TopicQueries around the same time, the period is 1 second, the number of samples per period is 10, the first TopicQuery selects 5 samples, and the second one selects 8, the DataWriter will immediately attempt to publish all 5 for the first TopicQuery and 5 for the second one. After one second, it will publish the remaining 3 samples.

Entity:

dds::pub::DataWriter (p. 1138)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

See also

dds::pub::Publisher (p. 1993)

8.393.2 Constructor & Destructor Documentation

8.393.2.1 TopicQueryDispatch() [1/2]

```
rti::core::policy::TopicQueryDispatch::TopicQueryDispatch ( )  [inline]
```

Creates the default policy.

8.393.2.2 TopicQueryDispatch() [2/2]

```
rti::core::policy::TopicQueryDispatch::TopicQueryDispatch (
            bool the_enable,
            const  dds::core::Duration & the_publication_period,
            int32_t the_samples_per_period )  [inline]
```

Creates a policy with the provided values for enable, publication_period and samples_per_period.

8.393.3 Member Function Documentation

8.393.3.1 enable() [1/2]

```
TopicQueryDispatch & rti::core::policy::TopicQueryDispatch::enable (
            bool the_enable )
```

<<***extension***>> *(*p. *116)* Allows this writer to dispatch TopicQueries

When set to true, this writer can receive and dispatch TopicQueries.

[default] false

Categories

Topic Queries (p. 239), **Immutable** (p. 243), **Discovery Metadata** (p. 288)

8.393.3.2 enable() [2/2]

```
bool rti::core::policy::TopicQueryDispatch::enable ( ) const
```

Getter (see setter with the same name)

8.393.3.3 publication_period() [1/2]

```
TopicQueryDispatch & rti::core::policy::TopicQueryDispatch::publication_period (
            const  dds::core::Duration & the_publication_period )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* Sets the periodic interval at which samples are published.

[default] 1 second

[range] [0,1 year]

Categories

Topic Queries (p. 239), **Immutable** (p. 243)

8.393.3.4 publication_period() [2/2]

```
dds::core::Duration rti::core::policy::TopicQueryDispatch::publication_period ( ) const
```

Getter (see setter with the same name)

8.393.3.5 samples_per_period() [1/2]

```
TopicQueryDispatch & rti::core::policy::TopicQueryDispatch::samples_per_period (
            int32_t the_samples_per_period )
```

<<***extension***>> *(*p. *116)* Sets the maximum number of samples to publish in each publication_period.

[default] dds::core::LENGTH_UNLIMITED (p. 438)

[range] [1, 100000000] or **dds::core::LENGTH_UNLIMITED** (p. 438)

Categories

Topic Queries (p. 239), **Immutable** (p. 243)

8.393.3.6 samples_per_period() [2/2]

```
int32_t rti::core::policy::TopicQueryDispatch::samples_per_period ( ) const
```

Getter (see setter with the same name)

8.394 rti::sub::TopicQuerySelection Class Reference

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Specifies the data query that defines a **TopicQuery** (p. 2516).

```
#include <rti/sub/TopicQuery.hpp>
```

Public Types

- typedef **TopicQuerySelectionKind** **Kind**
  Alias for TopicQuerySelectionKind.

Public Member Functions

- **TopicQuerySelection** (const **dds::topic::Filter** &the_filter)
  *Creates a **TopicQuerySelection** (p. 2526).*
- **TopicQuerySelection** (const **dds::topic::Filter** &the_filter, **Kind** the_kind)
  *Creates a **TopicQuerySelection** (p. 2526) with a selection kind.*
- **dds::topic::Filter** & **filter** ()
  Gets the filter.
- const **dds::topic::Filter** & **filter** () const
  Gets the filter.
- **Kind** **kind** () const
  Gets the selection kind.
- **TopicQuerySelection** & **kind** (**Kind** the_kind)
  Indicates whether the sample selection is limited to cached samples or not.

8.394.1 Detailed Description

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Specifies the data query that defines a **TopicQuery** (p. 2516).

A **TopicQuerySelection** (p. 2526) is defined by a **dds::topic::Filter** (p. 1558) that selects a subset of samples in the dds::pub::DataWriters

8.394.2 Member Typedef Documentation

8.394.2.1 Kind

```
typedef  TopicQuerySelectionKind  rti::sub::TopicQuerySelection::Kind
```

Alias for TopicQuerySelectionKind.

8.394.3 Constructor & Destructor Documentation

8.394.3.1 TopicQuerySelection() [1/2]

```
rti::sub::TopicQuerySelection::TopicQuerySelection (
            const  dds::topic::Filter & the_filter )  [inline], [explicit]
```

Creates a **TopicQuerySelection** (p. 2526).

Parameters

the_filter	Defines the data query

Note

The filter expression can start with the special condition "@instance_state = ALIVE AND" followed by the rest of the expression. This restricts the selection to samples of alive instances.

See also

TopicQuery::SelectAll() (p. 2519)

TopicQuery::UseReaderContentFilter() (p. 2518)

8.394.3.2 TopicQuerySelection() [2/2]

```
rti::sub::TopicQuerySelection::TopicQuerySelection (
            const  dds::topic::Filter & the_filter,
             Kind the_kind )  [inline]
```

Creates a **TopicQuerySelection** (p. 2526) with a selection kind.

Parameters

the_filter	Defines the data query
the_kind	Indicates the kind of selection

8.394.4 Member Function Documentation

8.394.4.1 filter() [1/2]

```
dds::topic::Filter & rti::sub::TopicQuerySelection::filter ( )  [inline]
```

Gets the filter.

8.394.4.2 filter() [2/2]

```
const  dds::topic::Filter & rti::sub::TopicQuerySelection::filter ( ) const  [inline]
```

Gets the filter.

8.394.4.3 kind() [1/2]

```
Kind rti::sub::TopicQuerySelection::kind ( ) const  [inline]
```

Gets the selection kind.

8.394.4.4 kind() [2/2]

```
TopicQuerySelection & rti::sub::TopicQuerySelection::kind (
            Kind the_kind )  [inline]
```

Indicates whether the sample selection is limited to cached samples or not.

[default] rti::sub::TopicQuerySelectionKind::HISTORY_SNAPSHOT

8.395 rti::sub::TopicQuerySelectionKind_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) **rti::sub::TopicQuerySelectionKind** (p. 71).

```
#include <TopicQuery.hpp>
```

Public Types

- enum **type** {
  HISTORY_SNAPSHOT ,
  CONTINUOUS }
  The underlying `enum` *type.*

8.395.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) **rti::sub::TopicQuerySelectionKind** (p. 71).

8.395.2 Member Enumeration Documentation

8.395.2.1 type

`enum` **rti::sub::TopicQuerySelectionKind_def::type**

The underlying `enum` type.

Enumerator

HISTORY_SNAPSHOT	Indicates that the **rti::sub::TopicQuery** (p. 2516) may only select samples that were in the DataWriter cache upon reception. **[default]**
CONTINUOUS	Indicates that the **rti::sub::TopicQuery** (p. 2516) may continue selecting samples published after its reception. The subscribing application must explicitly delete the **TopicQuery** (p. 2516) (see **rti::sub::TopicQuery::close()** (p. 2518)) to signal the DataWriters to stop publishing samples for it.

8.396 rti::util::network_capture::TrafficKindMask Class Reference

<<***extension***>> *(*p. *354)* Mask indicating the traffic direction to capture.

`#include <network_capture.hpp>`

Inherits std::bitset< 32 >.

Public Types

- typedef std::bitset< 32 > **MaskType**
  A typedef of std::bitset<32> for convenience.

Public Member Functions

- **TrafficKindMask** ()

  *Default constructor for **TrafficKindMask*** (p. *2529*).
- **TrafficKindMask** (uint64_t mask)

  *Construct a **TrafficKindMask*** (p. *2529*) *from an integer.*
- **TrafficKindMask** (const **MaskType** &mask)

  *Construct a **TrafficKindMask*** (p. *2529*) *from a MaskType object.*

Static Public Member Functions

- static const **TrafficKindMask** **out** ()

  Do not capture outbound traffic.
- static const **TrafficKindMask** **in** ()

  Do not capture inbound traffic.
- static const **TrafficKindMask** **default_mask** ()

  *Default mask for **network_capture::TrafficKindMask*** (p. *2529*).
- static const **TrafficKindMask** **none** ()

  Do not capture any traffic.
- static const **TrafficKindMask** **all** ()

  Capture all traffic (both inbound and outbound).

8.396.1 Detailed Description

<<***extension***>> *(p. 354)* Mask indicating the traffic direction to capture.

The masks are based on a combination (or only one) of the **network_capture::TrafficKindMask** (p. 2529) bitmaps.

See also

network_capture::TrafficKindMask (p. 2529)

8.396.2 Member Typedef Documentation

8.396.2.1 MaskType

`typedef std::bitset<32>` **`rti::util::network_capture::TrafficKindMask::MaskType`**

A typedef of std::bitset<32> for convenience.

8.396.3 Constructor & Destructor Documentation

8.396.3.1 TrafficKindMask() [1/3]

```
rti::util::network_capture::TrafficKindMask::TrafficKindMask ( )  [inline]
```

Default constructor for **TrafficKindMask** (p. 2529).

8.396.3.2 TrafficKindMask() [2/3]

```
rti::util::network_capture::TrafficKindMask::TrafficKindMask (
            uint64_t mask )  [inline], [explicit]
```

Construct a **TrafficKindMask** (p. 2529) from an integer.

Parameters

mask	Value whose bits are copied to the bitset positions

8.396.3.3 TrafficKindMask() [3/3]

```
rti::util::network_capture::TrafficKindMask::TrafficKindMask (
            const  MaskType & mask )  [inline]
```

Construct a **TrafficKindMask** (p. 2529) from a MaskType object.

Parameters

mask	A std::bitset<32> to construct this **TrafficKindMask** (p. 2529) from.

8.396.4 Member Function Documentation

8.396.4.1 out()

```
static const TrafficKindMask rti::util::network_capture::TrafficKindMask::out ( ) [inline],
[static]
```

Do not capture outbound traffic.

8.396.4.2 in()

```
static const TrafficKindMask rti::util::network_capture::TrafficKindMask::in ( ) [inline], [static]
```

Do not capture inbound traffic.

8.396.4.3 default_mask()

```
static const TrafficKindMask rti::util::network_capture::TrafficKindMask::default_mask ( ) [inline],
[static]
```

Default mask for **network_capture::TrafficKindMask** (p. 2529).

It is equivalent to **network_capture::TrafficKindMask::all()** (p. 2532).

[default] Capture all traffic: inbound and outbound.

8.396.4.4 none()

```
static const TrafficKindMask rti::util::network_capture::TrafficKindMask::none ( ) [inline],
[static]
```

Do not capture any traffic.

8.396.4.5 all()

```
static const TrafficKindMask rti::util::network_capture::TrafficKindMask::all ( ) [inline],
[static]
```

Capture all traffic (both inbound and outbound).

The value is equal to setting both the input and output bits of the mask: (**network_capture::TrafficKindMask::out()** (p. 2531) | **network_capture::TrafficKindMask::in()** (p. 2532)).

8.397 TransportAllocationSettings_t Struct Reference

Allocation settings used by various internal buffers.

`#include <transport_common_user.h>`

8.397.1 Detailed Description

Allocation settings used by various internal buffers.

An allocation setting structure defines the rules of memory management used by internal buffers.

An internal buffer can provide blocks of memory of fixed size. They are used in several places of any transport, and this structure defines its starting size, limits, and how to increase its capacity.

It contains three values:

- initial_count: the number of individual elements that are allocated when the buffer is created.
- max_count: the maximum number of elements the buffer can hold. The buffer will grow up to this amount. After this limit is reached, new allocation requests will fail. For unlimited size, use the value NDDS_TRANSPORT_↩ ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED
- incremental_count: The amount of elements that are allocated at every increment. You can use the value↩ : NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC to have the buffer double its size at every reallocation request.

8.398 rti::core::policy::TransportBuiltin Class Reference

<<***extension***>> *(*p. *116)* Specifies which built-in transports to use

`#include <rti/core/policy/CorePolicy.hpp>`

Public Member Functions

- **TransportBuiltin** ()

  Creates the default policy.
- **TransportBuiltin** (const **rti::core::policy::TransportBuiltinMask** &the_mask)

  Creates an instnace with the transport selection that the mask specifies.
- **TransportBuiltin** & **mask** (**rti::core::policy::TransportBuiltinMask** the_mask)

  Set the selected transports through a mask.
- **rti::core::policy::TransportBuiltinMask mask** () const

  Get the mask that indicates which transports are selected.

Static Public Member Functions

- static **TransportBuiltin All** ()

  *Creates a policy that selects **TransportBuiltinMask::all()** (p. 2539)*
- static **TransportBuiltin None** ()

  *Creates a policy that selects **TransportBuiltinMask::none()** (p. 2539)*
- static **TransportBuiltin Shmem** ()

  *Creates a policy that selects **TransportBuiltinMask::shmem()** (p. 2539)*
- static **TransportBuiltin UDPv4** ()

  *Creates a policy that selects **TransportBuiltinMask::udpv4()** (p. 2539)*
- static **TransportBuiltin UDPv6** ()

  *Creates a policy that selects **TransportBuiltinMask::udpv6()** (p. 2540)*
- static **TransportBuiltin UDPv4_WAN** ()

  *Creates a policy that selects **TransportBuiltinMask::udpv4_wan()** (p. 2540)*

Static Public Attributes

- static const std::string **SHMEM_ALIAS**

  Alias name for the shared-memory built-in transport: "builtin.shmem".
- static const std::string **UDPv4_ALIAS**

  Alias name for the UDPv4 built-in transport: "builtin.udpv4".
- static const std::string **UDPv4_WAN_ALIAS**

  Alias name for the UDPv4 asymmetric built-in transport: "builtin.udpv4_wan".
- static const std::string **UDPv6_ALIAS**

  Alias name for the UDPv6 built-in transport: "builtin.udpv6".

8.398.1 Detailed Description

<<***extension***>> *(p. 116)* Specifies which built-in transports to use

Three different transport plug-ins are built into the core RTI Connext libraries (for most supported target platforms): UDPv4, shared memory, and UDPv6.

This QoS policy allows you to control which of these built-in transport plug-ins are used by a **dds::domain::DomainParticipant** (p. 1326). By default, only the UDPv4 and shared memory plug-ins are enabled (although on some embedded platforms, the shared memory plug-in is not available). In some cases, users will disable the shared memory transport when they do not want applications to use shared memory to communicate when running on the same node.

Entity:

dds::domain::DomainParticipant (p. 1326)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.398.2 Constructor & Destructor Documentation

8.398.2.1 TransportBuiltin() [1/2]

```
rti::core::policy::TransportBuiltin::TransportBuiltin ( )  [inline]
```

Creates the default policy.

8.398.2.2 TransportBuiltin() [2/2]

```
rti::core::policy::TransportBuiltin::TransportBuiltin (
            const  rti::core::policy::TransportBuiltinMask & the_mask )  [inline]
```

Creates an instnace with the transport selection that the mask specifies.

8.398.3 Member Function Documentation

8.398.3.1 All()

```
static  TransportBuiltin rti::core::policy::TransportBuiltin::All ( )  [inline], [static]
```

Creates a policy that selects **TransportBuiltinMask::all()** (p. 2539)

8.398.3.2 None()

```
static  TransportBuiltin rti::core::policy::TransportBuiltin::None ( )  [inline], [static]
```

Creates a policy that selects **TransportBuiltinMask::none()** (p. 2539)

8.398.3.3 Shmem()

```
static  TransportBuiltin rti::core::policy::TransportBuiltin::Shmem ( )  [inline], [static]
```

Creates a policy that selects **TransportBuiltinMask::shmem()** (p. 2539)

8.398.3.4 UDPv4()

```
static TransportBuiltin rti::core::policy::TransportBuiltin::UDPv4 ( ) [inline], [static]
```

Creates a policy that selects **TransportBuiltinMask::udpv4()** (p. 2539)

8.398.3.5 UDPv6()

```
static TransportBuiltin rti::core::policy::TransportBuiltin::UDPv6 ( ) [inline], [static]
```

Creates a policy that selects **TransportBuiltinMask::udpv6()** (p. 2540)

8.398.3.6 UDPv4_WAN()

```
static TransportBuiltin rti::core::policy::TransportBuiltin::UDPv4_WAN ( ) [inline], [static]
```

Creates a policy that selects **TransportBuiltinMask::udpv4_wan()** (p. 2540)

8.398.3.7 mask() [1/2]

```
TransportBuiltin & rti::core::policy::TransportBuiltin::mask (
            rti::core::policy::TransportBuiltinMask the_mask )
```

Set the selected transports through a mask.

8.398.3.8 mask() [2/2]

```
rti::core::policy::TransportBuiltinMask rti::core::policy::TransportBuiltin::mask ( ) const
```

Get the mask that indicates which transports are selected.

References **rti::core::policy::TransportBuiltinMask::shmem()**.

8.398.4 Member Data Documentation

8.398.4.1 SHMEM_ALIAS

```
const std::string rti::core::policy::TransportBuiltin::SHMEM_ALIAS  [static]
```

Alias name for the shared-memory built-in transport: "builtin.shmem".

8.398.4.2 UDPv4_ALIAS

```
const std::string rti::core::policy::TransportBuiltin::UDPv4_ALIAS  [static]
```

Alias name for the UDPv4 built-in transport: "builtin.udpv4".

8.398.4.3 UDPv4_WAN_ALIAS

```
const std::string rti::core::policy::TransportBuiltin::UDPv4_WAN_ALIAS  [static]
```

Alias name for the UDPv4 asymmetric built-in transport: "builtin.udpv4_wan".

8.398.4.4 UDPv6_ALIAS

```
const std::string rti::core::policy::TransportBuiltin::UDPv6_ALIAS  [static]
```

Alias name for the UDPv6 built-in transport: "builtin.udpv6".

8.399 rti::core::policy::TransportBuiltinMask Class Reference

<<***extension***>> *(*p. *354)* Mask that specifies which built-in transports are used

```
#include <PolicyKind.hpp>
```

Inherits std::bitset< 8 >.

Public Types

- typedef std::bitset< 8 > **MaskType**

  The base type, std::bitset.

Public Member Functions

- **TransportBuiltinMask** (uint64_t mask)

  Creates a mask from the bits in an integer.
- **TransportBuiltinMask** (const **MaskType** &mask)

  Creates a mask from a std::bitset.

Static Public Member Functions

- static const **TransportBuiltinMask** **all** ()

  All bits are set.
- static const **TransportBuiltinMask** **none** ()

  No bits are set.
- static const **TransportBuiltinMask** **shmem** ()

  Selects the built-in shared-memory transport.
- static const **TransportBuiltinMask** **udpv4** ()

  Selects the built-in UDPv4 transport.
- static const **TransportBuiltinMask** **udpv4_wan** ()

  Selects the built-in UDPv4 asymmetric transport.
- static const **TransportBuiltinMask** **udpv6** ()

  Selects the built-in UDPv6 transport.

8.399.1 Detailed Description

<<***extension***>> *(*p. *354)* Mask that specifies which built-in transports are used

8.399.2 Member Typedef Documentation

8.399.2.1 MaskType

`typedef std::bitset<8>` **`rti::core::policy::TransportBuiltinMask::MaskType`**

The base type, std::bitset.

8.399.3 Constructor & Destructor Documentation

8.399.3.1 TransportBuiltinMask() [1/2]

```
rti::core::policy::TransportBuiltinMask::TransportBuiltinMask (
            uint64_t mask )  [inline], [explicit]
```

Creates a mask from the bits in an integer.

8.399.3.2 TransportBuiltinMask() [2/2]

```
rti::core::policy::TransportBuiltinMask::TransportBuiltinMask (
            const  MaskType & mask )  [inline]
```

Creates a mask from a std::bitset.

8.399.4 Member Function Documentation

8.399.4.1 all()

```
static const  TransportBuiltinMask rti::core::policy::TransportBuiltinMask::all ( )  [inline],
[static]
```

All bits are set.

8.399.4.2 none()

```
static const  TransportBuiltinMask rti::core::policy::TransportBuiltinMask::none ( )  [inline],
[static]
```

No bits are set.

8.399.4.3 shmem()

```
static const  TransportBuiltinMask rti::core::policy::TransportBuiltinMask::shmem ( )  [inline],
[static]
```

Selects the built-in shared-memory transport.

Referenced by **rti::core::policy::TransportBuiltin::mask()**.

8.399.4.4 udpv4()

```
static const TransportBuiltinMask rti::core::policy::TransportBuiltinMask::udpv4 ( ) [inline],
[static]
```

Selects the built-in UDPv4 transport.

8.399.4.5 udpv4_wan()

```
static const TransportBuiltinMask rti::core::policy::TransportBuiltinMask::udpv4_wan ( ) [inline],
[static]
```

Selects the built-in UDPv4 asymmetric transport.

8.399.4.6 udpv6()

```
static const TransportBuiltinMask rti::core::policy::TransportBuiltinMask::udpv6 ( ) [inline],
[static]
```

Selects the built-in UDPv6 transport.

Referenced by **rti::core::policy::TransportUnicast::TransportUnicast()**.

8.400 rti::core::TransportClassId_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) TransportClassId.

```
#include <TransportInfo.hpp>
```

Public Types

- enum **type** {
  INVALID = NDDS_TRANSPORT_CLASSID_INVALID ,
  ANY = NDDS_TRANSPORT_CLASSID_ANY ,
  UDPv4 = NDDS_TRANSPORT_CLASSID_UDPv4 ,
  UDPv4_WAN = NDDS_TRANSPORT_CLASSID_UDPv4_WAN ,
  SHMEM = NDDS_TRANSPORT_CLASSID_SHMEM ,
  SHMEM_510 = NDDS_TRANSPORT_CLASSID_SHMEM_510 ,
  INTRA = NDDS_TRANSPORT_CLASSID_INTRA ,
  UDPv6 = NDDS_TRANSPORT_CLASSID_UDPv6 ,
  UDPv6_510 =NDDS_TRANSPORT_CLASSID_UDPv6_510 ,
  TCPV4_LAN = NDDS_TRANSPORT_CLASSID_TCPV4_LAN ,
  TCPV4_WAN = NDDS_TRANSPORT_CLASSID_TCPV4_WAN ,
  TLSV4_LAN = NDDS_TRANSPORT_CLASSID_TLSV4_LAN ,
  TLSV4_WAN = NDDS_TRANSPORT_CLASSID_TLSV4_WAN ,
  RESERVED_RANGE = NDDS_TRANSPORT_CLASSID_RESERVED_RANGE }

  The underlying `enum` *type.*

8.400.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) TransportClassId.

8.400.2 Member Enumeration Documentation

8.400.2.1 type

`enum` **rti::core::TransportClassId_def::type**

The underlying `enum` type.

Enumerator

INVALID	Invalid.
UDPv4	UDPv4.
UDPv4_WAN	UDPv4 WAN.
SHMEM	Shared memory.
SHMEM_510	Shared memory in RTI Connext 5.1.0 and earlier.
UDPv6	UDPv6.
UDPv6_510	UDPv6 in RTI Connext 5.1.0 and earlier.
TCPV4_LAN	TCPv4 LAN mode.
TCPV4_WAN	TCPv4 WAN mode.
TLSV4_LAN	TCPv4 LAN mode with TLS.
TLSV4_WAN	TCPv4 WAN mode with TLS.
RESERVED_RANGE	Reserved for additional user-defined transport plugins.

8.401 rti::core::TransportInfo Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Contains the class id and message max size of an installed transport

`#include <rti/core/TransportInfo.hpp>`

Public Member Functions

- **TransportInfo** ()

  Creates an instance with default values.
- **TransportInfo** (const rti::core::TransportClassId::type &the_class_id, int32_t the_message_size_max)

  Creates an instance with the specified class id and max size.

- **rti::core::TransportClassId class_id** () const

  Getter (see setter with the same name)
- **TransportInfo** & **class_id** (const rti::core::TransportClassId::type &the_class_id)

  The class_id identifies the transport associated with the message_size_max.
- int32_t **message_size_max** () const

  Getter (see setter with the same name)
- **TransportInfo** & **message_size_max** (int32_t the_message_size_max)

  The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin identified by the class_id.

8.401.1 Detailed Description

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Contains the class id and message max size of an installed transport

8.401.2 Constructor & Destructor Documentation

8.401.2.1 TransportInfo() [1/2]

```
rti::core::TransportInfo::TransportInfo ( )  [inline]
```

Creates an instance with default values.

8.401.2.2 TransportInfo() [2/2]

```
rti::core::TransportInfo::TransportInfo (
            const rti::core::TransportClassId::type & the_class_id,
            int32_t the_message_size_max )  [inline]
```

Creates an instance with the specified class id and max size.

8.401.3 Member Function Documentation

8.401.3.1 class_id() [1/2]

```
rti::core::TransportClassId rti::core::TransportInfo::class_id ( ) const  [inline]
```

Getter (see setter with the same name)

8.401.3.2 class_id() [2/2]

```
TransportInfo & rti::core::TransportInfo::class_id (
            const rti::core::TransportClassId::type & the_class_id )  [inline]
```

The class_id identifies the transport associated with the message_size_max.

8.401.3.3 message_size_max() [1/2]

```
int32_t rti::core::TransportInfo::message_size_max ( ) const  [inline]
```

Getter (see setter with the same name)

8.401.3.4 message_size_max() [2/2]

```
TransportInfo & rti::core::TransportInfo::message_size_max (
            int32_t the_message_size_max )  [inline]
```

The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin identified by the class_id.

8.402 rti::core::policy::TransportMulticast Class Reference

<<***extension***>> *(p. 116)* Specifies the multicast address on which a **dds::sub::DataReader** (p. 976) wants to receive its data and other settings.

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **TransportMulticast** ()

  Creates the default policy.
- **TransportMulticast** (const **TransportMulticastSettingsSeq** &the_value, **TransportMulticastKind** the_kind)

  Creates an instance with the speicfied multicast settings.
- **TransportMulticast** & **settings** (const **TransportMulticastSettingsSeq** &the_settings)

  *<<**basic**>> (p. 104) <<**extension**>> (p. 116) A sequence of multicast communications settings.*
- **TransportMulticastSettingsSeq settings** () const

  Getter (see setter with the same name)
- **TransportMulticast** & **kind** (**TransportMulticastKind** the_kind)

  *<<**extension**>> (p. 116) A value that specifies a way to determine how to obtain the multicast address.*
- **TransportMulticastKind kind** () const

  Getter (see setter with the same name)

8.402.1 Detailed Description

*<<**extension**>> (p. 116)* Specifies the multicast address on which a **dds::sub::DataReader** (p. 976) wants to receive its data and other settings.

By default, a **dds::pub::DataWriter** (p. 1138) will send individually addressed packets for each **dds::sub::DataReader** (p. 976) that subscribes to the topic of the DataWriter – this is known as unicast delivery. Thus, as many copies of the data will be sent over the network as there are DataReaders for the data. The network bandwidth used by a DataWriter will thus increase linearly with the number of DataReaders.

Multicast addressing (on UDP/IP transports) allows multiple DataReaders to receive the *same* network packet. By using multicast, a **dds::pub::DataWriter** (p. 1138) can send a single network packet that is received by all subscribing applications. Thus the network bandwidth usage will be constant, independent of the number of DataReaders.

Coordinating the multicast address specified by DataReaders can help optimize network bandwidth usage in systems where there are multiple DataReaders for the same **dds::topic::Topic** (p. 2474).

Entity:

dds::sub::DataReader (p. 976)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.402.2 Constructor & Destructor Documentation

8.402.2.1 TransportMulticast() [1/2]

```
rti::core::policy::TransportMulticast::TransportMulticast ( )  [inline]
```

Creates the default policy.

8.402.2.2 TransportMulticast() [2/2]

```
rti::core::policy::TransportMulticast::TransportMulticast (
             const  TransportMulticastSettingsSeq & the_value,
              TransportMulticastKind the_kind )
```

Creates an instance with the speicfied multicast settings.

See individual setters.

8.402.3 Member Function Documentation

8.402.3.1 settings() [1/2]

```
 TransportMulticast & rti::core::policy::TransportMulticast::settings (
             const  TransportMulticastSettingsSeq & the_settings )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* A sequence of multicast communications settings.

An empty sequence means that multicast is not used by the entity.

The RTPS wire protocol currently limits the maximum number of multicast locators to

1. Note that this is a hard limit that cannot be increased. However, this limit can be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **rti::core::policy::Property** (p. 1969) associated with the **dds::domain::qos::DomainParticipantQos** (p. 1384).

[default] Empty sequence.

Categories

Immutable (p. 243), **Transports** (p. 237), **Discovery Metadata** (p. 288)

8.402.3.2 settings() [2/2]

```
TransportMulticastSettingsSeq rti::core::policy::TransportMulticast::settings ( ) const
```

Getter (see setter with the same name)

8.402.3.3 kind() [1/2]

```
TransportMulticast & rti::core::policy::TransportMulticast::kind (
            TransportMulticastKind the_kind )
```

<<***extension***>> *(*p. *116)* A value that specifies a way to determine how to obtain the multicast address.

This field can be set to one of the following two values: **rti::core::policy::TransportMulticastKind_def::AUTOMATIC** (p. 2547) or **rti::core::policy::TransportMulticastKind_def::UNICAST** (p. 2547).

- If it is set to **rti::core::policy::TransportMulticastKind_def::AUTOMATIC** (p. 2547), the behavior will depend on the **rti::core::policy::TransportMulticast::settings** (p. 2545):
  - If **rti::core::policy::TransportMulticast::settings** (p. 2545) does not have any elements, then multicast will not be used.
  - If **rti::core::policy::TransportMulticast::settings** (p. 2545) first element has an empty address, then the address will be obtained from **rti::core::policy::TransportMulticastMapping** (p. 2547).
  - If none of the elements in **rti::core::policy::TransportMulticast::settings** (p. 2545) is empty, and at least one element has a valid address, then that address will be used.
- If it is set to **rti::core::policy::TransportMulticastKind_def::UNICAST** (p. 2547), then multicast will not be used.

[default] rti::core::policy::TransportMulticastKind_def::AUTOMATIC (p. 2547)

Categories

Immutable (p. 243), **Transports** (p. 237), **Data Availability** (p. 167)

8.402.3.4 kind() [2/2]

```
TransportMulticastKind rti::core::policy::TransportMulticast::kind ( ) const
```

Getter (see setter with the same name)

8.403 rti::core::policy::TransportMulticastKind_def Struct Reference

<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) TransportMulticastKind

`#include <PolicyKind.hpp>`

Public Types

- enum **type** {
  AUTOMATIC ,
  UNICAST }

  The underlying `enum` *type.*

8.403.1 Detailed Description

<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) TransportMulticastKind

8.403.2 Member Enumeration Documentation

8.403.2.1 type

`enum` **rti::core::policy::TransportMulticastKind_def::type**

The underlying `enum` type.

Enumerator

AUTOMATIC	Selects the multicast address automatically. NOTE: This setting is required when using the **rti::core::policy::TransportMulticastMapping** (p. 2547).
UNICAST	Selects a unicast-only mode.

8.404 rti::core::policy::TransportMulticastMapping Class Reference

<<***extension***>> *(p. 116)* Specifies a list of topic_expressions and multicast addresses that can be used by an Entity with a specific topic name to receive data.

`#include <CorePolicy.hpp>`

Public Member Functions

- **TransportMulticastMapping** & **mappings** (const MulticastMappingSeq &the_mappings)

  *<<**extension**>> (p. 116) A sequence of multicast communication mappings.*
- MulticastMappingSeq **mappings** () const

  Getter (see setter with the same name)

8.404.1 Detailed Description

*<<**extension**>> (p. 116)* Specifies a list of topic_expressions and multicast addresses that can be used by an Entity with a specific topic name to receive data.

This QoS policy provides an alternate way to assign multicast receive addresses to DataReaders. It allows you to perform multicast configuration at the **dds::domain::DomainParticipant** (p. 1326) level.

To use multicast communication *without* this QoS policy, you must explicitly assign a multicast receive address on each **dds::sub::DataReader** (p. 976). This can quickly become difficult to configure as more DataReaders of different topics and multicast addresses are added.

With this QoS policy, you can configure a set of multicast addresses on the **dds::domain::DomainParticipant** (p. 1326); those addresses will then be automatically assigned to the DomainParticipant's DataReaders. A single configuration on the **dds::domain::DomainParticipant** (p. 1326) can thus replace per-DataReader configuration.

On the DomainParticipant, the set of assignable addresses can be configured for specific topics. Addresses are configured on topics because efficient usage of multicast will have all DataWriters and DataReaders of a single topic using the same multicast address.

You can specify a mapping between a topic's name and a multicast address. For example, topic 'A' can be assigned to address 239.255.1.1 and topic 'B' can be assigned to address 239.255.1.2.

You can use filter expressions to configure a subset of topics to use a specific list of addresses. For example, suppose topics "X", "Y" and "Z" need to be sent to any address within the range [239.255.1.1, 239.255.1.255]. You can specify an expression on the topic name (e.g. "[X-Z]") corresponding to that range of addresses. Then the DomainParticipant will select an address for a topic whose name matches the expression.

The middleware will use a hash function to perform the mapping from topic to address. Alternatively, you can specify a pluggable mapping function.

IMPORTANT: All the strings defined in each element of the sequence must be assigned using RTI_String_dup("foo");. For example:

mcastMappingElement->addresses = DDS_String_dup("[239.255.1.1,239.255.1.255]");

NOTE: To use this QoS policy, you must:

- Set **rti::core::policy::TransportMulticast::kind** (p. 2546) to **rti::core::policy::TransportMulticastKind_def↩ ::AUTOMATIC** (p. 2547).
- Set the first element in **dds::sub::DataReader** (p. 976)'s **rti::core::policy::TransportMulticast::settings** (p. 2545) with an empty address.

Entity:

dds::domain::DomainParticipant (p. 1326)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.404.2 Member Function Documentation

8.404.2.1 mappings() [1/2]

```
TransportMulticastMapping & rti::core::policy::TransportMulticastMapping::mappings (
            const MulticastMappingSeq & the_mappings )
```

<<***extension***>> *(*p. *116)* A sequence of multicast communication mappings.

[default] Empty sequence.

Categories

Immutable (p. 243), **Transports** (p. 237), **Data Availability** (p. 167)

8.404.2.2 mappings() [2/2]

```
MulticastMappingSeq rti::core::policy::TransportMulticastMapping::mappings ( ) const
```

Getter (see setter with the same name)

8.405 rti::core::TransportMulticastSettings Class Reference

<<***extension***>> *(*p. *354)* Represents a list of multicast locators

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **TransportMulticastSettings** (const **dds::core::StringSeq** &the_transports, const std::string &the_receive_↩ address, int32_t the_receive_port)

  Creates an instance with the specified transport aliases, receive address and receive port.
- **TransportMulticastSettings** & **transports** (const **dds::core::StringSeq** &the_transports)

  <<***extension***>> *(*p. *116) A sequence of transport aliases that specifies the transports on which to receive multicast traffic for the entity.*
- **dds::core::StringSeq transports** () const

  Getter (see the setter with the same name)
- **TransportMulticastSettings** & **receive_address** (const std::string &the_receive_address)

  <<***extension***>> *(*p. *116) The multicast group address on which the entity can receive data.*
- std::string **receive_address** () const

  Getter (see the setter with the same name)
- **TransportMulticastSettings** & **receive_port** (int32_t the_receive_port)

  <<***extension***>> *(*p. *116) The multicast port on which the entity can receive data.*
- int32_t **receive_port** () const

  Getter (see the setter with the same name)

8.405.1 Detailed Description

<<***extension***>> *(*p. *354)* Represents a list of multicast locators

A multicast locator specifies a transport class, a multicast address, and a multicast port number on which messages can be received by an entity.

QoS:

rti::core::policy::TransportMulticast (p. 2543)

8.405.2 Constructor & Destructor Documentation

8.405.2.1 TransportMulticastSettings()

```
rti::core::TransportMulticastSettings::TransportMulticastSettings (
            const  dds::core::StringSeq & the_transports,
            const std::string & the_receive_address,
            int32_t the_receive_port )  [inline]
```

Creates an instance with the specified transport aliases, receive address and receive port.

See individual setters.

8.405.3 Member Function Documentation

8.405.3.1 transports() [1/2]

```
TransportMulticastSettings & rti::core::TransportMulticastSettings::transports (
            const  dds::core::StringSeq & the_transports )  [inline]
```

<<***extension***>> *(*p. *116)* A sequence of transport aliases that specifies the transports on which to receive *multicast* traffic for the entity.

Of the transport instances available to the entity, only those with aliases matching an alias in this sequence are used to subscribe to the multicast group addresses. Thus, this list of aliases sub-selects from the transport s available to the entity.

An empty sequence is a special value that specifies all the transports available to the entity.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 548).

[default] Empty sequence; i.e. all the transports available to the entity.

[range] Any sequence of non-null, non-empty strings.

8.405.3.2 transports() [2/2]

```
dds::core::StringSeq rti::core::TransportMulticastSettings::transports ( ) const  [inline]
```

Getter (see the setter with the same name)

8.405.3.3 receive_address() [1/2]

```
TransportMulticastSettings & rti::core::TransportMulticastSettings::receive_address (
            const std::string & the_receive_address )
```

<<***extension***>> *(*p. *116)* The multicast group address on which the entity can receive data.

Must must be an address in the proper format (see **Address Format** (p. 561)).

[default] NONE/INVALID. Required to specify a multicast group address to join.

[range] A valid IPv4 or IPv6 multicast address.

See also

Address Format (p. 561)

8.405.3.4 receive_address() [2/2]

```
std::string rti::core::TransportMulticastSettings::receive_address ( ) const  [inline]
```

Getter (see the setter with the same name)

8.405.3.5 receive_port() [1/2]

```
TransportMulticastSettings & rti::core::TransportMulticastSettings::receive_port (
            int32_t the_receive_port )  [inline]
```

<<***extension***>> *(*p. *116)* The multicast port on which the entity can receive data.

[default] 0, which implies that the actual port number is determined by a formula as a function of the `domain_id` (see **rti::core::policy::WireProtocol::participant_id** (p. 2635)).

[range] [0,0xffffffff]

8.405.3.6 receive_port() [2/2]

```
int32_t rti::core::TransportMulticastSettings::receive_port ( ) const  [inline]
```

Getter (see the setter with the same name)

8.406 dds::core::policy::TransportPriority Class Reference

Allows applications to take advantage of transports capable of sending messages with different priorities.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **TransportPriority** ()
  Creates a policy with priority 0.
- **TransportPriority** (int32_t priority)
  Creates a policy with the specified priority.
- **TransportPriority** & **value** (int32_t priority)
  Sets the priority.
- int32_t **value** () const
  Getter (see setter with the same name)

8.406.1 Detailed Description

Allows applications to take advantage of transports capable of sending messages with different priorities.

The Transport Priority QoS policy is optional and only supported on certain OSs and transports. It allows you to specify on a per-DataWriter or DataReader basis that the data sent by that endpoint is of a different priority.

The DDS specification does not indicate how a DDS implementation should treat data of different priorities. It is often difficult or impossible for DDS implementations to treat data of higher priority differently than data of lower priority, especially when data is being sent (delivered to a physical transport) directly by the thread that called **dds::pub::Data↩ Writer::write()** (p. 1148). Also, many physical network transports themselves do not have a end-user controllable level of data packet priority.

Entity:

dds::pub::DataWriter (p. 1138), **dds::sub::DataReader** (p. 976), **dds::topic::Topic** (p. 2474)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.406.2 Usage

In RTI Connext, for the IP-based transports (UDPv4, UDPv6, Real-Time WAN, and TCP), the value set in the Transport Priority QoS policy can be used to set the differentiated services field (DS field) bits of the IPv4 and IPv6 headers for datagrams sent by a **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976). It is platform-dependent on how and whether setting the DS field has an effect. Some platforms may require external permissions in order to set the DS field.

The transport priority value is not provided as is to the transports, but transformed according to the following fields: transport_priority_mask (for example, see **NDDS_Transport_UDPv4_Property_t::transport_priority_mask** (p. 1804)), transport_priority_mapping_low (for example, see **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_low** (p. 1805)), and transport_priority_mapping_high (for example, see **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_high** (p. 1805)). If you want the priority value to be exactly equal to the DS value, then the only change you need to make is to set transport_priority_mask to `0xff` (and keep the transport_priority_mapping_low and transport_priority_mapping_high defaults).

It is incorrect to assume that using the Transport Priority QoS policy will have any effect at all on the end-to-end delivery of data from a **dds::pub::DataWriter** (p. 1138) and a **dds::sub::DataReader** (p. 976). All network elements, including switches and routers, must have the capability and be enabled to actually use the DS field to treat higher priority packets differently. Thus the ability to use the Transport Priority QoS policy must be designed and configured at a *system* level; just turning it on in an application may have no effect at all at a transport level.

For additional details on how to set the DS field in IP-based transports, see the "TRANSPORT_PRIORITY QosPolicy" section of the `User's Manual`.

8.406.3 Constructor & Destructor Documentation

8.406.3.1 TransportPriority() [1/2]

```
dds::core::policy::TransportPriority::TransportPriority ( )  [inline]
```

Creates a policy with priority 0.

8.406.3.2 TransportPriority() [2/2]

```
dds::core::policy::TransportPriority::TransportPriority (
            int32_t priority )  [inline], [explicit]
```

Creates a policy with the specified priority.

8.406.4 Member Function Documentation

8.406.4.1 value() [1/2]

```
TransportPriority & dds::core::policy::TransportPriority::value (
            int32_t priority )  [inline]
```

Sets the priority.

You may choose any value within the range of a 32-bit signed integer; higher values indicate higher priority. However, any further interpretation of this policy is specific to a particular transport and a particular DDS implementation. For example, a particular transport is permitted to treat a range of priority values as equivalent to one another.

[default] 0

Categories

Immutable (p. 243), **Transports** (p. 237), **Concurrency** (p. 164)

8.406.4.2 value() [2/2]

```
int32_t dds::core::policy::TransportPriority::value ( ) const  [inline]
```

Getter (see setter with the same name)

8.407 rti::core::policy::TransportSelection Class Reference

<<***extension***>> *(p. 116)* Specifies the transports that a **dds::pub::DataWriter** (p. 1138) or a **dds::sub::DataReader** (p. 976) may use to send or receive data

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **TransportSelection** ()

  Creates the default policy.
- **TransportSelection** (const **dds::core::StringSeq** &the_enabled_transports)

  Creates an instance with the sepcified transport aliases.
- **TransportSelection** & **enabled_transports** (const **dds::core::StringSeq** &the_enabled_transports)

  <<***extension***>> *(p. 116) A sequence of transport aliases that specifies the transport instances available for use by the entity.*
- **dds::core::StringSeq enabled_transports** () const

  Getter (see setter with the same name)

8.407.1 Detailed Description

<<***extension***>> *(*p. *116)* Specifies the transports that a **dds::pub::DataWriter** (p. 1138) or a **dds::sub::DataReader** (p. 976) may use to send or receive data

An application may be simultaneously connected to many different physical transports, e.g., Ethernet, Infiniband, shared memory, VME backplane, and wireless. By default, RTI Connext will use up to 16 transports to deliver data from a DataWriter to a DataReader.

This QoS policy can be used to both limit and control which of the application's available transports may be used by a **dds::pub::DataWriter** (p. 1138) to send data or by a **dds::sub::DataReader** (p. 976) to receive data.

Entity:

dds::sub::DataReader (p. 976), **dds::pub::DataWriter** (p. 1138)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.407.2 Constructor & Destructor Documentation

8.407.2.1 TransportSelection() [1/2]

```
rti::core::policy::TransportSelection::TransportSelection ( )  [inline]
```

Creates the default policy.

8.407.2.2 TransportSelection() [2/2]

```
rti::core::policy::TransportSelection::TransportSelection (
            const  dds::core::StringSeq & the_enabled_transports )  [inline], [explicit]
```

Creates an instance with the sepcified transport aliases.

8.407.3 Member Function Documentation

8.407.3.1 enabled_transports() [1/2]

```
TransportSelection & rti::core::policy::TransportSelection::enabled_transports (
            const  dds::core::StringSeq & the_enabled_transports )
```

<<***extension***>> *(*p. *116)* A sequence of transport aliases that specifies the transport instances available for use by the entity.

Of the transport instances installed with the **dds::domain::DomainParticipant** (p. 1326), only those with aliases matching an alias in this sequence are available to the entity.

Thus, this list of aliases sub-selects from the transports available to the **dds::domain::DomainParticipant** (p. 1326).

An empty sequence is a special value that specifies all the transports installed with the **dds::domain::Domain↩ Participant** (p. 1326).

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 548). These alias names are case sensitive and should be written in lowercase.

[default] Empty sequence; i.e. all the transports installed with and available to the **dds::domain::DomainParticipant** (p. 1326).

[range] A sequence of non-empty strings.

See also

dds::domain::qos::DomainParticipantQos::transport_builtin.

Categories

Immutable (p. 243), **Transports** (p. 237)

8.407.3.2 enabled_transports() [2/2]

```
dds::core::StringSeq rti::core::policy::TransportSelection::enabled_transports ( ) const
```

Getter (see setter with the same name)

8.408 rti::core::policy::TransportUnicast Class Reference

<<***extension***>> *(*p. *116)* Specifies a subset of transports and a port number that can be used by a **dds::core::Entity** (p. 1520) to receive data.

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **TransportUnicast** ()
  Creates the default policy.
- **TransportUnicast** (const **rti::core::TransportUnicastSettingsSeq** & **settings**)
  Creates an instance with the specified settings.
- **TransportUnicast** & **settings** (const **rti::core::TransportUnicastSettingsSeq** &value)
  Sets the unicast settings.
- **rti::core::TransportUnicastSettingsSeq settings** () const
  Gets the unicast settings.

8.408.1 Detailed Description

<<***extension***>> *(p. 116)* Specifies a subset of transports and a port number that can be used by a **dds::core::Entity** (p. 1520) to receive data.

Entity:

dds::domain::DomainParticipant (p. 1326), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.408.2 Usage

RTI Connext may send data to a variety of Entities, not just DataReaders. For example, reliable DataWriters may receive ACK/NACK packets from reliable DataReaders.

During discovery, each **dds::core::Entity** (p. 1520) announces to remote applications a list of (up to 16) unicast addresses to which the remote application should send data (either user data packets or reliable protocol meta-data such as ACK/NACKs and heartbeats). Note that this is a hard limit that cannot be increased. However, this limit can be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **rti::core::policy::Property** (p. 1969) associated with the **dds::domain::qos::DomainParticipantQos** (p. 1384).

Note: RTI Connext internally uses several dynamic locator lists to keep track of configured unicast locators. The initial size of these lists can be configured with the **dds::domain::DomainParticipant** (p. 1326) property 'dds.domain_participant.initial_locator_list_size'. By default, the initial size is derived from local information, such as the number of local locators. To prevent dynamic resizing at runtime, set the property's value to the maximum number of locators that a single participant or endpoint may advertise at any given time in your system.

By default, the list of addresses is populated automatically with values obtained from the enabled transport plug-ins allowed to be used by the Entity (see **rti::core::policy::TransportBuiltin** (p. 2533) and **rti::core::policy::TransportSelection** (p. 2554)). Also, the associated ports are automatically determined (see **rti::core::RtpsWellKnownPorts** (p. 2258)).

Use this QoS policy to manually set the receive address list for an Entity. You may optionally set a port to use a non-default receive port as well. Only the first 16 addresses will be used.

RTI Connext will create a receive thread for every unique port number that it encounters (on a per transport basis).

- For a **dds::domain::DomainParticipant** (p. 1326), this QoS policy sets the default list of addresses used by other applications to send user data for local DataReaders.

- For a **dds::sub::DataReader** (p. 976), if set, then other applications will use the specified list of addresses to send user data (and reliable protocol packets for reliable DataReaders). Otherwise, if not set, the other applications will use the addresses set by the **dds::domain::DomainParticipant** (p. 1326).

- For a reliable **dds::pub::DataWriter** (p. 1138), if set, then other applications will use the specified list of addresses to send reliable protocol packets (ACKS/NACKS) on the behalf of reliable DataReaders. Otherwise, if not set, the other applications will use the addresses set by the **dds::domain::DomainParticipant** (p. 1326).

8.408.3 Constructor & Destructor Documentation

8.408.3.1 TransportUnicast() [1/2]

```
rti::core::policy::TransportUnicast::TransportUnicast ( )  [inline]
```

Creates the default policy.

References **rti::core::policy::TransportBuiltinMask::udpv6()**.

8.408.3.2 TransportUnicast() [2/2]

```
rti::core::policy::TransportUnicast::TransportUnicast (
            const  rti::core::TransportUnicastSettingsSeq & settings )  [explicit]
```

Creates an instance with the specified settings.

8.408.4 Member Function Documentation

8.408.4.1 settings() [1/2]

```
TransportUnicast & rti::core::policy::TransportUnicast::settings (
            const  rti::core::TransportUnicastSettingsSeq & value )
```

Sets the unicast settings.

An empty sequence means that applicable defaults specified by elsewhere (e.g. dds::domain::qos::DomainParticipant↩ Qos::default_unicast) should be used.

The RTPS wire protocol currently limits the maximum number of unicast locators to

1. Note that this is a hard limit that cannot be increased. However, this limit can be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **rti::core::policy::Property** (p. 1969) associated with the **dds::domain::qos::DomainParticipantQos** (p. 1384).

[default] Empty sequence.

See also

dds::domain::qos::DomainParticipantQos::default_unicast

Categories

Immutable (p. 243), **Transports** (p. 237), **Discovery Metadata** (p. 288)

8.408.4.2 settings() [2/2]

```
rti::core::TransportUnicastSettingsSeq rti::core::policy::TransportUnicast::settings ( ) const
```

Gets the unicast settings.

8.409 rti::core::TransportUnicastSettings Class Reference

<<***extension***>> *(p. 354)* Represents a list of unicast locators

```
#include <rti/core/PolicySettings.hpp>
```

Public Member Functions

- **TransportUnicastSettings** ()

  Creates the default policy.
- **TransportUnicastSettings** (const **dds::core::StringSeq** & **transports**, int32_t **receive_port**=0)

  Creates an instance with the specified transports and receive_port.
- **TransportUnicastSettings** & **transports** (const **dds::core::StringSeq** &value)

  Sets a sequence of transport aliases that specifies the unicast interfaces on which to receive unicast traffic for the entity.
- **dds::core::StringSeq transports** () const

  Getter (see setter with the same name)
- **TransportUnicastSettings** & **receive_port** (int32_t value)

  The unicast port on which the entity can receive data.
- int32_t **receive_port** () const

  Getter (see setter with the same name)

8.409.1 Detailed Description

<<***extension***>> *(*p. *354)* Represents a list of unicast locators

A unicast locator specifies a transport class, a unicast address, and a unicast port number on which messages can be received by an entity.

QoS:

rti::core::policy::TransportUnicast (p. 2556)

8.409.2 Constructor & Destructor Documentation

8.409.2.1 TransportUnicastSettings() [1/2]

```
rti::core::TransportUnicastSettings::TransportUnicastSettings ( )  [inline]
```

Creates the default policy.

8.409.2.2 TransportUnicastSettings() [2/2]

```
rti::core::TransportUnicastSettings::TransportUnicastSettings (
            const  dds::core::StringSeq & transports,
            int32_t receive_port = 0 )  [explicit]
```

Creates an instance with the specified transports and receive_port.

8.409.3 Member Function Documentation

8.409.3.1 transports() [1/2]

```
TransportUnicastSettings & rti::core::TransportUnicastSettings::transports (
            const  dds::core::StringSeq & value )
```

Sets a sequence of transport aliases that specifies the unicast interfaces on which to receive *unicast* traffic for the entity.

Of the transport instances available to the entity, only those with aliases matching an alias on this sequence are used to determine the unicast interfaces used by the entity.

Thus, this list of aliases sub-selects from the transports available to the entity.

Each unicast interface on a transport results in a unicast locator for the entity.

An empty sequence is a special value that specifies all the transports available to the entity.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 548).

[default] Empty sequence; i.e. all the transports available to the entity.
[range] Any sequence of non-null, non-empty strings.

8.409.3.2 transports() [2/2]

```
dds::core::StringSeq rti::core::TransportUnicastSettings::transports ( ) const
```

Getter (see setter with the same name)

8.409.3.3 receive_port() [1/2]

```
TransportUnicastSettings & rti::core::TransportUnicastSettings::receive_port (
            int32_t value )
```

The unicast port on which the entity can receive data.

Must be an *unused* unicast port on the system.

[default] 0, which implies that the actual port number is determined by a formula as a function of the `domain_id`, and the **rti::core::policy::WireProtocol::participant_id** (p. 2635).

[range] [0,0xffffffff]

See also

rti::core::policy::WireProtocol::participant_id (p. 2635).

8.409.3.4 receive_port() [2/2]

```
int32_t rti::core::TransportUnicastSettings::receive_port ( ) const
```

Getter (see setter with the same name)

8.410 rti::topic::trust::TrustAlgorithmRequirements Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Type to describe Trust Plugins algorithm requirements.

```
#include <rti/topic/trust/TrustAlgorithmRequirements.hpp>
```

Public Member Functions

- **TrustAlgorithmRequirements** ()=default

  Create an instance of ***TrustAlgorithmRequirements*** *(p. 2562) with default values.*
- uint32_t **supported_mask** () const

  Get the Trust Plugins algorithms that an entity supports.
- uint32_t **required_mask** () const

  Get the Trust Plugins algorithms that an entity uses.

8.410.1 Detailed Description

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Type to describe Trust Plugins algorithm requirements.

8.410.2 Constructor & Destructor Documentation

8.410.2.1 TrustAlgorithmRequirements()

```
rti::topic::trust::TrustAlgorithmRequirements::TrustAlgorithmRequirements ( )  [default]
```

Create an instance of **TrustAlgorithmRequirements** (p. 2562) with default values.

The meaning of this field may vary depending on what Trust Plugins the entity is using.

8.410.3 Member Function Documentation

8.410.3.1 supported_mask()

```
uint32_t rti::topic::trust::TrustAlgorithmRequirements::supported_mask ( ) const  [inline]
```

Get the Trust Plugins algorithms that an entity supports.

8.410.3.2 required_mask()

```
uint32_t rti::topic::trust::TrustAlgorithmRequirements::required_mask ( ) const  [inline]
```

Get the Trust Plugins algorithms that an entity uses.

8.411 dds::core::policy::TypeConsistencyEnforcement Class Reference

Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **TypeConsistencyEnforcement** ()
  Creates the default policy.
- **TypeConsistencyEnforcement** (**dds::core::policy::TypeConsistencyEnforcementKind** the_kind)
  Creates an instance with a specific enforcement kind.
- **TypeConsistencyEnforcement** & **kind** (**dds::core::policy::TypeConsistencyEnforcementKind** value)
  Sets the enforcement kind.
- **dds::core::policy::TypeConsistencyEnforcementKind kind** () const
  Gets the enforcement kind.
- **TypeConsistencyEnforcement** & **ignore_sequence_bounds** (bool ignore)
  Controls whether sequence bounds are ignored.
- bool **ignore_sequence_bounds** () const
  Getter (see setter with the same name)
- **TypeConsistencyEnforcement** & **ignore_string_bounds** (bool ignore)
  Controls whether string bounds are ignored.
- bool **ignore_string_bounds** () const
  Getter (see setter with the same name)
- **TypeConsistencyEnforcement** & **ignore_member_names** (bool ignore)
  Controls whether member names are ignored.
- bool **ignore_member_names** () const
  Getter (see setter with the same name)
- **TypeConsistencyEnforcement** & **prevent_type_widening** (bool prevent)
  Controls whether type widening is prevented.

- bool **prevent_type_widening** () const

  Getter (see setter with the same name)
- **TypeConsistencyEnforcement** & **force_type_validation** (bool force)

  Controls whether type validation is forced.
- bool **force_type_validation** () const

  Getter (see setter with the same name)
- **TypeConsistencyEnforcement** & **ignore_enum_literal_names** (bool ignore)

  Controls whether enumeration literal names are ignored.
- bool **ignore_enum_literal_names** () const

  Getter (see setter with the same name)

Static Public Member Functions

- static **TypeConsistencyEnforcement** **AllowTypeCoercion** ()

  Creates an instance with TypeConsistencyEnforcementKind::ALLOW_TYPE_COERCION.
- static **TypeConsistencyEnforcement** **DisallowTypeCoercion** ()

  Creates an instance with TypeConsistencyEnforcementKind::DISALLOW_TYPE_COERCION.
- static **TypeConsistencyEnforcement** **AutoTypeCoercion** ()

  Creates an instance with TypeConsistencyEnforcementKind::AUTO_TYPE_COERCION.

8.411.1 Detailed Description

Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

This policy defines a type consistency kind, which allows applications to select from among a set of predetermined behaviors. The following consistency kinds are specified: TypeConsistencyKind::DISALLOW_TYPE_COERCION, TypeConsistencyKind::ALLOW_TYPE_COERCION and TypeConsistencyKind::AUTO_TYPE_COERCION.

The type-consistency-enforcement rules consist of two steps:

Step 1. If both the DataWriter and DataReader specify a TypeObject, it is considered first. If the DataReader allows type coercion, then its type must be assignable from the DataWriter's type, taking into account the values of prevent_type_widening, ignore_sequence_bounds, ignore_string_bounds, ignore_member_names, and ignore_enum_literal_names. If the DataReader does not allow type coercion, then its type must be equivalent to the type of the DataWriter.

Step 2. If either the DataWriter or the DataReader does not provide a TypeObject definition, then the registered type names are examined. The DataReader's and DataWriter's registered type names must match exactly, as was true in RTI Connext releases prior to 5.0.0.

If either Step 1 or Step 2 fails, the Topics associated with the DataReader and DataWriter are considered to be inconsistent and the **dds::core::status::InconsistentTopicStatus** (p. 1611) is updated.

The default enforcement kind is TypeConsistencyKind::AUTO_TYPE_COERCION. This default kind translates to TypeConsistencyKind::ALLOW_TYPE_COERCION except in the following cases:

- When a **Zero Copy** (p. 50) DataReader is used, the kind is translated to TypeConsistencyKind::DISALLOW_TYPE_COERCION.

- When the middleware is introspecting the built-in topic data declaration of a remote DataReader in order to determine whether it can match with a local DataWriter, if it observes that no TypeConsistencyEnforcement↩ QosPolicy value is provided (as would be the case when communicating with a Service implementation not in conformance with this specification), it assumes a kind of TypeConsistencyKind::DISALLOW_TYPE_COERCION.

For additional information on type consistency enforcement refer to the `Extensible Types Guide` and the `OMG Extensible and Dynamic Topic Types for DDS Specification`.

Entity:

dds::sub::DataReader (p. 976)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

8.411.2 Constructor & Destructor Documentation

8.411.2.1 TypeConsistencyEnforcement() [1/2]

```
dds::core::policy::TypeConsistencyEnforcement::TypeConsistencyEnforcement ( )  [inline]
```

Creates the default policy.

8.411.2.2 TypeConsistencyEnforcement() [2/2]

```
dds::core::policy::TypeConsistencyEnforcement::TypeConsistencyEnforcement (
            dds::core::policy::TypeConsistencyEnforcementKind the_kind )  [inline], [explicit]
```

Creates an instance with a specific enforcement kind.

8.411.3 Member Function Documentation

8.411.3.1 kind() [1/2]

```
TypeConsistencyEnforcement & dds::core::policy::TypeConsistencyEnforcement::kind (
            dds::core::policy::TypeConsistencyEnforcementKind value ) [inline]
```

Sets the enforcement kind.

[default] TypeConsistencyKind::AUTO_TYPE_COERCION

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Type System** (p. 240)

8.411.3.2 kind() [2/2]

```
dds::core::policy::TypeConsistencyEnforcementKind dds::core::policy::TypeConsistencyEnforcement↩
::kind ( ) const [inline]
```

Gets the enforcement kind.

8.411.3.3 ignore_sequence_bounds() [1/2]

```
TypeConsistencyEnforcement & dds::core::policy::TypeConsistencyEnforcement::ignore_sequence_↩
bounds (
            bool ignore ) [inline]
```

Controls whether sequence bounds are ignored.

If the option is set to true, then sequence bounds (maximum lengths) are not considered as part of the type assignability. This means that a T2 sequence type with maximum length L2 would be assignable to a T1 sequence type with maximum length L1, even if L2 is greater than L1. If the option is set to false, then sequence bounds are taken into consideration for type assignability, and in order for T1 to be assignable from T2, it is required that L1 >= L2.

[default] true

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Type System** (p. 240)

8.411.3.4 ignore_sequence_bounds() [2/2]

```
bool dds::core::policy::TypeConsistencyEnforcement::ignore_sequence_bounds ( ) const  [inline]
```

Getter (see setter with the same name)

8.411.3.5 ignore_string_bounds() [1/2]

```
TypeConsistencyEnforcement & dds::core::policy::TypeConsistencyEnforcement::ignore_string_bounds
(
            bool ignore )  [inline]
```

Controls whether string bounds are ignored.

If the option is set to true, then string bounds (maximum lengths) are not considered as part of the type assignability. This means that a T2 string type with maximum length L2 would be assignable to a T1 string type with maximum length L1, even if L2 is greater than L1. If the option is set to false, then string bounds are taken into consideration for type assignability, and in order for T1 to be assignable from T2, it is required that L1 $>=$ L2.

[default] true

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Type System** (p. 240)

8.411.3.6 ignore_string_bounds() [2/2]

```
bool dds::core::policy::TypeConsistencyEnforcement::ignore_string_bounds ( ) const  [inline]
```

Getter (see setter with the same name)

8.411.3.7 ignore_member_names() [1/2]

```
TypeConsistencyEnforcement & dds::core::policy::TypeConsistencyEnforcement::ignore_member_names (
            bool ignore )  [inline]
```

Controls whether member names are ignored.

If the option is set to true, then member names are not considered as part of the type assignability. If the option is set to false, then member names are taken into consideration for type assignability, and in order for members with the same ID to be assignable, the members must also have the same name.

[default] false

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Type System** (p. 240)

8.411.3.8 ignore_member_names() [2/2]

```
bool dds::core::policy::TypeConsistencyEnforcement::ignore_member_names ( ) const  [inline]
```

Getter (see setter with the same name)

8.411.3.9 prevent_type_widening() [1/2]

```
TypeConsistencyEnforcement & dds::core::policy::TypeConsistencyEnforcement::prevent_type_widening
(
            bool prevent )  [inline]
```

Controls whether type widening is prevented.

If the option is set to false, then type widening is permitted. If the option is set to true, then a wider type may not be assignable from a narrower type.

[default] false

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Type System** (p. 240)

8.411.3.10 prevent_type_widening() [2/2]

```
bool dds::core::policy::TypeConsistencyEnforcement::prevent_type_widening ( ) const  [inline]
```

Getter (see setter with the same name)

8.411.3.11 force_type_validation() [1/2]

```
TypeConsistencyEnforcement & dds::core::policy::TypeConsistencyEnforcement::force_type_validation
(
            bool force )  [inline]
```

Controls whether type validation is forced.

If the option is set to true, then type information must be available in order to complete matching between a **dds::pub::DataWriter** (p. 1138) and a **dds::sub::DataReader** (p. 976). If the option is set to false, then matching can occur without complete type information as long as the type names match exactly. Note that if the types have the same name but are not assignable, DataReaders may fail to deserialize incoming data samples.

[default] false

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Type System** (p. 240)

8.411.3.12 force_type_validation() [2/2]

```
bool dds::core::policy::TypeConsistencyEnforcement::force_type_validation ( ) const  [inline]
```

Getter (see setter with the same name)

8.411.3.13 ignore_enum_literal_names() [1/2]

```
 TypeConsistencyEnforcement & dds::core::policy::TypeConsistencyEnforcement::ignore_enum_literal↩
_names (
             bool ignore )  [inline]
```

Controls whether enumeration literal names are ignored.

If the option is set to true, then enumeration constants may change their names, but not their values, and still maintain assignability. If the option is set to false, then in order for enumerations to be assignable, any constant that has the same value in both enumerations must also have the same name.

[default] false

Categories

Immutable (p. 243), **Matching** (p. 293), **Discovery Metadata** (p. 288), **Type System** (p. 240)

8.411.3.14 ignore_enum_literal_names() [2/2]

```
bool dds::core::policy::TypeConsistencyEnforcement::ignore_enum_literal_names ( ) const  [inline]
```

Getter (see setter with the same name)

8.411.3.15 AllowTypeCoercion()

```
static  TypeConsistencyEnforcement dds::core::policy::TypeConsistencyEnforcement::AllowType↩
Coercion ( )  [inline], [static]
```

Creates an instance with TypeConsistencyEnforcementKind::ALLOW_TYPE_COERCION.

8.411.3.16 DisallowTypeCoercion()

```
static TypeConsistencyEnforcement dds::core::policy::TypeConsistencyEnforcement::DisallowType↩
Coercion ( ) [inline], [static]
```

Creates an instance with TypeConsistencyEnforcementKind::DISALLOW_TYPE_COERCION.

8.411.3.17 AutoTypeCoercion()

```
static TypeConsistencyEnforcement dds::core::policy::TypeConsistencyEnforcement::AutoTypeCoercion
( ) [inline], [static]
```

Creates an instance with TypeConsistencyEnforcementKind::AUTO_TYPE_COERCION.

8.412 dds::core::policy::TypeConsistencyEnforcementKind_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) TypeConsistencyEnforcementKind.

```
#include <PolicyKind.hpp>
```

Public Types

- enum **type** {
  DISALLOW_TYPE_COERCION ,
  ALLOW_TYPE_COERCION ,
  AUTO_TYPE_COERCION }
  The underlying `enum` *type.*

8.412.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) TypeConsistencyEnforcementKind.

8.412.2 Member Enumeration Documentation

8.412.2.1 type

```
enum dds::core::policy::TypeConsistencyEnforcementKind_def::type
```

The underlying `enum` type.

Enumerator

DISALLOW_TYPE_COERCION	The DataWriter and the DataReader must support the same data type in order for them to communicate. This is the degree of type consistency enforcement required by the `OMG DDS Specification` prior to the `OMG Extensible and Dynamic Topic Types for DDS Specification`.

ALLOW_TYPE_COERCION	The DataWriter and the DataReader need not support the same data type in order for them to communicate as long as the DataReader's type is assignable from the DataWriter's type. For example, the following two extensible types will be assignable to each other since MyDerivedType contains all the members of MyBaseType (member_1) plus some additional elements (member_2).
	`struct MyBaseType { long member_1; };`
	`struct MyDerivedType: MyBaseType { long member_2; };`
	Even if MyDerivedType was not explicitly inheriting from MyBaseType the types would still be assignable. For example:
	`struct MyBaseType { long member_1; };`
	`struct MyDerivedType { long member_1; long member_2; };`
	For additional information on type assignability refer to the `OMG Extensible and Dynamic Topic Types for DDS Specification`.

AUTO_TYPE_COERCION	This AUTO value will be applied as TypeConsistencyKind::DISALLOW_TYPE_COERCION when the data type is annotated with @transfer_mode(SHMEM_REF) and the @language_binding is PLAIN (default) while using C, Traditional C++, or Modern C++ APIs. In all other cases, this AUTO value will be applied as TypeConsistencyKind::ALLOW_TYPE_COERCION. **[default]**

8.413 dds::core::xtypes::TypeKind_def Struct Reference

The definition of TypeKind.

`#include <TypeKind.hpp>`

Public Types

- enum **type** { }

  The underlying `enum` type.

8.413.1 Detailed Description

The definition of TypeKind.

8.413.2 Member Enumeration Documentation

8.413.2.1 type

enum **dds::core::xtypes::TypeKind_def::type**

The underlying enum type.

Enumerator

PRIMITIVE_TYPE	Flag indicating a primitive type. **See also** **is_primitive_type** (p. 633)
CONSTRUCTED_TYPE	Flag indicating a constructed type. **See also** **is_constructed_type** (p. 633)
COLLECTION_TYPE	Flag indicating a collection type. **See also** **is_collection_type** (p. 633)
AGGREGATION_TYPE	Flag indicating an aggreation type. **See also** **is_aggregation_type** (p. 634)
BOOLEAN_TYPE	**See also** **PrimitiveType** (p. 1959)
UINT_8_TYPE	**See also** **PrimitiveType** (p. 1959)
INT_16_TYPE	**See also** **PrimitiveType** (p. 1959)
UINT_16_TYPE	**See also** **PrimitiveType** (p. 1959)

Enumerator

INT_32_TYPE	**See also** **PrimitiveType** (p. 1959)
UINT_32_TYPE	**See also** **PrimitiveType** (p. 1959)
INT_64_TYPE	**See also** **PrimitiveType** (p. 1959)
UINT_64_TYPE	**See also** **PrimitiveType** (p. 1959)
FLOAT_32_TYPE	**See also** **PrimitiveType** (p. 1959)
FLOAT_64_TYPE	**See also** **PrimitiveType** (p. 1959)
FLOAT_128_TYPE	Indicates a PrimitiveType<rti::core::LongDouble> **See also** **PrimitiveType** (p. 1959)
CHAR_8_TYPE	**See also** **PrimitiveType** (p. 1959)
INT_8_TYPE	**See also** **PrimitiveType** (p. 1959)
OCTET_TYPE	**See also** **PrimitiveType** (p. 1959)
ENUMERATION_TYPE	Indicates an **EnumType** (p. 1535).
ALIAS_TYPE	Indicates an **AliasType** (p. 801).
ARRAY_TYPE	Indicates an **ArrayType** (p. 828).
SEQUENCE_TYPE	Indicates a **SequenceType** (p. 2344).

Enumerator

STRING_TYPE	Indicates a **StringType** (p. 2399).
WSTRING_TYPE	Indicates a **WStringType** (p. 2665).
UNION_TYPE	Indicates a **UnionType** (p. 2584).
STRUCTURE_TYPE	Indicates a **StructType** (p. 2400).

8.414 rti::core::policy::TypeSupport Class Reference

<<***extension***>> *(p. 116)* Allows attaching application-specific information to a **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976) that is passed to the serilization and deserialization routines.

`#include <rti/core/policy/CorePolicy.hpp>`

Public Member Functions

- **TypeSupport** ()

  Creates the default policy.
- **TypeSupport** & **plugin_data** (void ∗the_plugin_data)

  [DEPRECATED] *(p. **296**)* <<***extension***>> *(p. 116) Value to pass into the type plugin's de-/serialization function*
- void ∗ **plugin_data** ()

  Getter (see setter with the same name)
- **TypeSupport** & **cdr_padding_kind** (**CdrPaddingKind** the_cdr_padding_kind)

  [DEPRECATED] *(p. **296**)* <<***extension***>> *(p. 116) Determines whether or not the padding bytes will be set to zero during CDR serialization.*
- **CdrPaddingKind** **cdr_padding_kind** () const

  Getter (see setter with the same name)

8.414.1 Detailed Description

<<***extension***>> *(p. 116)* Allows attaching application-specific information to a **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976) that is passed to the serilization and deserialization routines.

The purpose of this QoS is to allow a user application to pass data to a type plugin's support functions and choose whether or not to set the padding bytes to zero when serializing a sample using CDR encapsulation.

Entity:

dds::domain::DomainParticipant (p. 1326), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Properties:

RxO (p. 514) = NO

Changeable (p. 514) = **UNTIL ENABLE** (p. 514)

8.414.2 Usage

The **rti::core::policy::TypeSupport::plugin_data** (p. 2575) allows you to associate a pointer to an object with a **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976). This object pointer is passed to the serialization routine of the data type associated with the **dds::pub::DataWriter** (p. 1138) or the deserialization routine of the data type associated with the **dds::sub::DataReader** (p. 976).

You can modify the rtiddsgen-generated code so that the de/serialization routines act differently depending on the information passed in via the object pointer. (The generated serialization and deserialization code does not use the pointer.)

This functionality can be used to change how data sent by a **dds::pub::DataWriter** (p. 1138) or received by a **dds::sub::DataReader** (p. 976) is serialized or deserialized on a per DataWriter and DataReader basis.

It can also be used to dynamically change how serialization (or for a less common case, deserialization) occurs. For example, a data type could represent a table, including the names of the rows and columns. However, since the row/column names of an instance of the table (a Topic) don't change, they only need to be sent once. The information passed in through the **TypeSupport** (p. 2574) QoS policy could be used to signal the serialization routine to send the row/column names the first time a **dds::pub::DataWriter** (p. 1138) calls **dds::pub::DataWriter::write()** (p. 1148), and then never again.

The **rti::core::policy::TypeSupport::cdr_padding_kind** (p. 2576) allows you to choose whether or not the padding bytes are set to zero during CDR serialization.

8.414.3 Constructor & Destructor Documentation

8.414.3.1 TypeSupport()

```
rti::core::policy::TypeSupport::TypeSupport ( )  [inline]
```

Creates the default policy.

8.414.4 Member Function Documentation

8.414.4.1 plugin_data() [1/2]

```
TypeSupport & rti::core::policy::TypeSupport::plugin_data (
            void * the_plugin_data )
```

[DEPRECATED] (p. **296)** <<***extension***>> *(*p. *116)* Value to pass into the type plugin's de-/serialization function

[default] NULL

Categories

Immutable (p. 243), **Application-Defined Metadata** (p. 159)

8.414.4.2 plugin_data() [2/2]

```
void * rti::core::policy::TypeSupport::plugin_data ( )
```

Getter (see setter with the same name)

8.414.4.3 cdr_padding_kind() [1/2]

```
TypeSupport & rti::core::policy::TypeSupport::cdr_padding_kind (
            CdrPaddingKind the_cdr_padding_kind )
```

[DEPRECATED] (p. **296**) <<***extension***>> *(p. 116)* Determines whether or not the padding bytes will be set to zero during CDR serialization.

In a DomainParticipant, this value configures how the padding bytes are set when serializing data for the Built-In Topic DataWriters and DataReaders. A value of CdrPaddingKind::AUTO_CDR_PADDING defaults to CdrPaddingKind::NOT↩ _SET_CDR_PADDING.

For DataWriters and DataReaders, this value configures how padding bytes are set when serializing data for that entity. A value of CdrPaddingKind::AUTO_CDR_PADDING means that the entity will inherit whatever value is set on the DomainParticipant.

[default] CdrPaddingKind::AUTO_CDR_PADDING

Categories

Immutable (p. 243), **Security** (p. 235)

8.414.4.4 cdr_padding_kind() [2/2]

```
CdrPaddingKind rti::core::policy::TypeSupport::cdr_padding_kind ( ) const
```

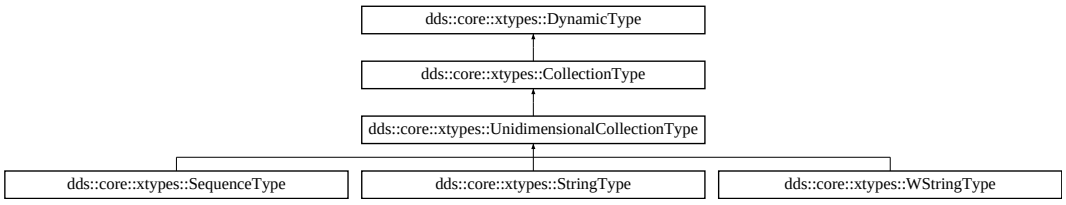
Getter (see setter with the same name)

8.415 dds::core::xtypes::UnidimensionalCollectionType Class Reference

<<***value-type***>> *(p. 350)* The base class of collection types with only one dimension.

```
#include <dds/core/xtypes/CollectionTypes.hpp>
```

Inheritance diagram for dds::core::xtypes::UnidimensionalCollectionType:



Public Member Functions

- uint32_t **bounds** () const

  Gets the maximum length of this collection.

Additional Inherited Members

8.415.1 Detailed Description

<<***value-type***>> *(*p. *350)* The base class of collection types with only one dimension.

8.415.2 Member Function Documentation

8.415.2.1 bounds()

```
uint32_t dds::core::xtypes::UnidimensionalCollectionType::bounds ( ) const
```

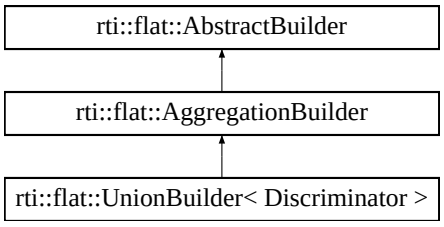
Gets the maximum length of this collection.

8.416 rti::flat::UnionBuilder< Discriminator > Class Template Reference

Base class of builders for user-defined mutable unions.

```
#include <AggregationBuilders.hpp>
```

Inheritance diagram for rti::flat::UnionBuilder< Discriminator >:



Additional Inherited Members

8.416.1 Detailed Description

template<**typename Discriminator**>
class rti::flat::UnionBuilder< **Discriminator** >

Base class of builders for user-defined mutable unions.

Union builders can only add or build a single member.

This class contains implementation details and doesn't add any public function to **AbstractBuilder** (p. 783).

8.417 dds::core::xtypes::UnionMember Class Reference

<<***value-type***>> *(p. 350)* Represents a **UnionType** (p. 2584) member

`#include "dds/core/xtypes/MemberType.hpp"`

Public Types

- typedef DDS_Long **DiscriminatorType**
  The type to represent a union discriminator.
- typedef std::vector< **DiscriminatorType** > **LabelSeq**
  A sequence of values for the discriminator that select a member.

Public Member Functions

- **UnionMember** (const std::string & **name**, const DynamicTypeImpl & **type**, const **LabelSeq** & **labels**)
  Creates a union member with a name, type and selected by one or more labels.
- **UnionMember** (const std::string & **name**, const DynamicTypeImpl & **type**, **DiscriminatorType** **label**)
  Creates a union member with a name, type and selected by a single label.
- **UnionMember** (const std::string &the_name, DynamicTypeImpl &&the_type, const **LabelSeq** &the_labels)
  <<***C++11***>> *(p. 353) Creates a union member with a name, type and selected by one or more labels.*
- **UnionMember** (const std::string &the_name, DynamicTypeImpl &&the_type, **DiscriminatorType** the_label)
  <<***C++11***>> *(p. 353) Creates a union member with a name, type and selected by a single label.*
- const **dds::core::string** & **name** () const
  Gets the member name.
- **dds::core::string** & **name** ()
  Gets the member name.
- const **DynamicType** & **type** ()
  Gets the member type.
- bool **has_id** () const
  Indicates it the member has an ID annotation.

- int32_t **get_id** () const

  Returns the ID annotation of this member.
- bool **is_pointer** () const

  *<<**extension**>> (p. 354) Indicates if this member is a pointer*
- uint32_t **label_count** () const

  Gets the number of labels that select this member.
- **LabelSeq** **labels** () const

  Gets a copy of all the labels that select this member.
- **UnionMember** & **labels** (const **LabelSeq** &values)

  Sets the labels that select this member.
- **UnionMember** & **label** (**DiscriminatorType** value)

  Sets a single label that selects this member.
- **UnionMember** & **name** (const **dds::core::string** &value)

  Sets the name.
- **UnionMember** & **id** (int32_t value)

  Sets the ID annotation of a member.
- **UnionMember** & **pointer** (bool value)

  *<<**extension**>> (p. 354) Makes a member a pointer*

Static Public Attributes

- static OMG_DDS_API_CLASS_VARIABLE **DiscriminatorType** **DEFAULT_LABEL**

  Special value for the `default:` union label.
- static const int32_t **INVALID_ID**

  The special ID of a member without the ID annotation.

8.417.1 Detailed Description

*<<**value-type**>> (p. 350)* Represents a **UnionType** (p. 2584) member

Encapsulates the name, type and labels of a **UnionType** (p. 2584) member along with several IDL annotations such as id.

8.417.2 Member Typedef Documentation

8.417.2.1 DiscriminatorType

```
typedef DDS_Long dds::core::xtypes::UnionMember::DiscriminatorType
```

The type to represent a union discriminator.

8.417.2.2 LabelSeq

```
typedef std::vector< DiscriminatorType> dds::core::xtypes::UnionMember::LabelSeq
```

A sequence of values for the discriminator that select a member.

8.417.3 Constructor & Destructor Documentation

8.417.3.1 UnionMember() [1/4]

```
dds::core::xtypes::UnionMember::UnionMember (
            const std::string & name,
            const DynamicTypeImpl & type,
            const  LabelSeq & labels )
```

Creates a union member with a name, type and selected by one or more labels.

8.417.3.2 UnionMember() [2/4]

```
dds::core::xtypes::UnionMember::UnionMember (
            const std::string & name,
            const DynamicTypeImpl & type,
             DiscriminatorType label )
```

Creates a union member with a name, type and selected by a single label.

8.417.3.3 UnionMember() [3/4]

```
dds::core::xtypes::UnionMember::UnionMember (
            const std::string & the_name,
            DynamicTypeImpl && the_type,
            const  LabelSeq & the_labels )  [inline]
```

<<***C++11***>> *(*p. *353)* Creates a union member with a name, type and selected by one or more labels.

It moves the type.

8.417.3.4 UnionMember() [4/4]

```
dds::core::xtypes::UnionMember::UnionMember (
            const std::string & the_name,
            DynamicTypeImpl && the_type,
             DiscriminatorType the_label )  [inline]
```

<<***C++11***>> *(p. 353)* Creates a union member with a name, type and selected by a single label.

It moves the type.

8.417.4 Member Function Documentation

8.417.4.1 name() [1/3]

```
const  dds::core::string & dds::core::xtypes::UnionMember::name ( ) const
```

Gets the member name.

8.417.4.2 name() [2/3]

```
 dds::core::string & dds::core::xtypes::UnionMember::name ( )
```

Gets the member name.

8.417.4.3 type()

```
const  DynamicType & dds::core::xtypes::UnionMember::type ( )
```

Gets the member type.

8.417.4.4 has_id()

```
bool dds::core::xtypes::UnionMember::has_id ( ) const
```

Indicates it the member has an ID annotation.

Returns

True if this member has been annotated explicitly with an ID; false if the has a default-assigned ID.

8.417.4.5 get_id()

```
int32_t dds::core::xtypes::UnionMember::get_id ( ) const
```

Returns the ID annotation of this member.

Returns

If **has_id()** (p. 2581) is true, this returns the value of the ID annotation. Otherwise it returns INVALID_ID.

8.417.4.6 is_pointer()

```
bool dds::core::xtypes::UnionMember::is_pointer ( ) const
```

<<***extension***>> *(p. 354)* Indicates if this member is a pointer

8.417.4.7 label_count()

```
uint32_t dds::core::xtypes::UnionMember::label_count ( ) const
```

Gets the number of labels that select this member.

8.417.4.8 labels() [1/2]

```
LabelSeq dds::core::xtypes::UnionMember::labels ( ) const
```

Gets a copy of all the labels that select this member.

8.417.4.9 labels() [2/2]

```
UnionMember & dds::core::xtypes::UnionMember::labels (
            const  LabelSeq & values )
```

Sets the labels that select this member.

8.417.4.10 label()

```
UnionMember & dds::core::xtypes::UnionMember::label (
             DiscriminatorType value )
```

Sets a single label that selects this member.

If more than one label selects this case, use **labels(const LabelSeq&)** (p. 2582)

8.417.4.11 name() [3/3]

```
UnionMember & dds::core::xtypes::UnionMember::name (
            const  dds::core::string & value )
```

Sets the name.

8.417.4.12 id()

```
UnionMember & dds::core::xtypes::UnionMember::id (
            int32_t value )
```

Sets the ID annotation of a member.

[default] Automatically assigned as the ID of the previous member plus one previous member

8.417.4.13 pointer()

```
UnionMember & dds::core::xtypes::UnionMember::pointer (
            bool value )
```

<<***extension***>> *(p. 354)* Makes a member a pointer

[default] false

8.417.5 Member Data Documentation

8.417.5.1 DEFAULT_LABEL

```
OMG_DDS_API_CLASS_VARIABLE  DiscriminatorType dds::core::xtypes::UnionMember::DEFAULT_LABEL  [static]
```

Special value for the `default:` union label.

8.417.5.2 INVALID_ID

```
const int32_t dds::core::xtypes::UnionMember::INVALID_ID  [static]
```

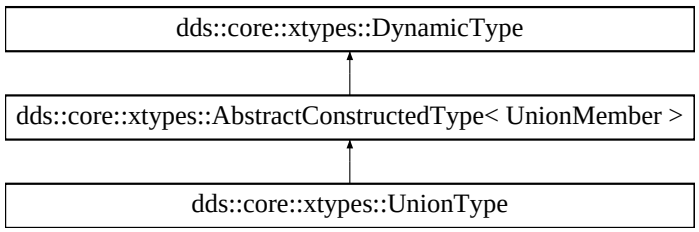
The special ID of a member without the ID annotation.

8.418 dds::core::xtypes::UnionType Class Reference

<<***value-type***>> *(*p. *350)* Represents an IDL `union` type

```
#include <dds/core/xtypes/UnionType.hpp>
```

Inheritance diagram for dds::core::xtypes::UnionType:



Public Types

- typedef UnionMember::DiscriminatorType **DiscriminatorType**

  The type used to set and get discriminator (label) values.

Public Member Functions

- **UnionType** (const std::string &the_name, const DynamicTypeImpl &discriminator_type)

  Creates an empty union.
- template<typename Container >
  UnionType (const std::string &the_name, const DynamicTypeImpl &discriminator_type, const Container &the_↩ members)

  Creates a union with the members in a container.
- template<typename MemberIter >
  UnionType (const std::string &the_name, const DynamicTypeImpl &discriminator_type, MemberIter begin, MemberIter end)

  Creates a union with the members in an iterator range.
- **UnionType** (const std::string &the_name, const DynamicTypeImpl &discriminator_type, std::initializer_list< **UnionMember** > the_members)

  Creates a union with the members in an initializer_list.
- const DynamicTypeImpl & **discriminator** () const

  Gets the type of the discriminator.
- **MemberIndex find_member_by_label** (**DiscriminatorType** label) const

Gets the index of the member selected by a label.

- **MemberIndex find_member_by_id** (uint32_t id) const

  Gets the index of the member with a specific member ID.

- **UnionType** & **add_member** (const **UnionMember** & **member**)

  Add a member to the end of this union.

- template<typename Container >
  UnionType & **add_members** (const Container &the_members)

  Adds all the members of a container at the end.

- **UnionType** & **add_members** (std::initializer_list< **UnionMember** > the_members)

  Adds all the members of an initializer_list at the end.

- template<typename MemberIter >
  UnionType & **add_members** (MemberIter begin, MemberIter end)

  Adds all the members in an iterator range at the end.

- **UnionType** & **add_member** (**UnionMember** &&the_member)

  Adds a member, moving it, at the end.

Additional Inherited Members

8.418.1 Detailed Description

<<***value-type***>> *(*p. *350)* Represents an IDL `union` type

Examples

Foo.hpp.

8.418.2 Member Typedef Documentation

8.418.2.1 DiscriminatorType

```
typedef UnionMember::DiscriminatorType dds::core::xtypes::UnionType::DiscriminatorType
```

The type used to set and get discriminator (label) values.

Note that this is different from **discriminator()** (p. 2587), which represents the discriminator type as defined in IDL.

8.418.3 Constructor & Destructor Documentation

8.418.3.1 UnionType() [1/4]

```
dds::core::xtypes::UnionType::UnionType (
            const std::string & the_name,
            const DynamicTypeImpl & discriminator_type )  [inline]
```

Creates an empty union.

Members can be added after creation.

Parameters

the_name	The name of the type
discriminator_type	The type of the discriminator

8.418.3.2 UnionType() [2/4]

```
template<typename Container >
dds::core::xtypes::UnionType::UnionType (
            const std::string & the_name,
            const DynamicTypeImpl & discriminator_type,
            const Container & the_members )  [inline]
```

Creates a union with the members in a container.

Template Parameters

Container	A container that provides the member functions **begin()** (p. 1669) and **end()** (p. 1670) to iterate over **UnionMember** (p. 2578) elements.

Parameters

the_name	The name of the type
discriminator_type	The type of the discriminator, which can be a primitive type, an **EnumType** (p. 1535) or an **AliasType** (p. 801) to any of these.
the_members	A container with the members for this union type

8.418.3.3 UnionType() [3/4]

```
template<typename MemberIter >
dds::core::xtypes::UnionType::UnionType (
            const std::string & the_name,
            const DynamicTypeImpl & discriminator_type,
            MemberIter begin,
            MemberIter end )  [inline]
```

Creates a union with the members in an iterator range.

Template Parameters

MemberIter	A forward iterator of **UnionMember** (p. 2578) elements

Parameters

the_name	The name of the type
discriminator_type	The type of the discriminator
begin	The beginning of the range of UnionMembers
end	The end of the range of UnionMembers

8.418.3.4 UnionType() [4/4]

```
dds::core::xtypes::UnionType::UnionType (
            const std::string & the_name,
            const DynamicTypeImpl & discriminator_type,
            std::initializer_list< UnionMember > the_members )  [inline]
```

Creates a union with the members in an initializer_list.

Parameters

the_name	The name of the type
discriminator_type	The type of the discriminator
the_members	An ininitalizer_list of UnionMembers

8.418.4 Member Function Documentation

8.418.4.1 discriminator()

```
const DynamicTypeImpl & dds::core::xtypes::UnionType::discriminator ( ) const
```

Gets the type of the discriminator.

8.418.4.2 find_member_by_label()

```
MemberIndex dds::core::xtypes::UnionType::find_member_by_label (
            DiscriminatorType label ) const
```

Gets the index of the member selected by a label.

The result of this function can be passed to member(uint32_t).

Returns

The index of the member selected by label or INVALID_INDEX if that label doesn't select any member

8.418.4.3 find_member_by_id()

```
MemberIndex dds::core::xtypes::UnionType::find_member_by_id (
            uint32_t id ) const
```

Gets the index of the member with a specific member ID.

The result of this function can be passed to member(uint32_t).

Returns

The index of the member that has this id or INVALID_INDEX if no member uses that id.

8.418.4.4 add_member() [1/2]

```
UnionType & dds::core::xtypes::UnionType::add_member (
            const  UnionMember & member )
```

Add a member to the end of this union.

8.418.4.5 add_members() [1/3]

```
template<typename Container >
UnionType & dds::core::xtypes::UnionType::add_members (
            const Container & the_members )  [inline]
```

Adds all the members of a container at the end.

8.418.4.6 add_members() [2/3]

```
UnionType & dds::core::xtypes::UnionType::add_members (
            std::initializer_list<  UnionMember > the_members )  [inline]
```

Adds all the members of an initializer_list at the end.

8.418.4.7 add_members() [3/3]

```
template<typename MemberIter >
UnionType & dds::core::xtypes::UnionType::add_members (
            MemberIter begin,
            MemberIter end )  [inline]
```

Adds all the members in an iterator range at the end.

8.418.4.8 add_member() [2/2]

```
 UnionType & dds::core::xtypes::UnionType::add_member (
             UnionMember && the_member )  [inline]
```

Adds a member, moving it, at the end.

8.419 rti::core::UnregisterThreadOnExit Class Reference

<<***extension***>> *(*p. *354)* Utility that calls **rti::core::unregister_thread** (p. 437) when leaving scope

```
#include <rti/core/thread.hpp>
```

Public Member Functions

- ~**UnregisterThreadOnExit** ()

  *Calls **rti::core::unregister_thread** (*p. *437).*

8.419.1 Detailed Description

<<***extension***>> *(*p. *354)* Utility that calls **rti::core::unregister_thread** (p. 437) when leaving scope

Declare an instance of this type in a thread function before any DDS API is called. After the function exits, this variable will be destroyed and the thread unregistered after everything else. This is only required on platforms that do not support automatic thread-specific storage (TSS) cleanup. Refer to the `Platform Notes` to see if your platform supports automatic TSS cleanup. It will be mentioned in the 'Thread Configuration' sub-section of the platform you are using.

See `Unregistering Threads`, in the Core Libraries User's Manual, for the equivalent API call in each of the supported language bindings.

For example:

```
void my_thread()
{
    // Whenever my_thread leaves, rti::core::unregister_thread will be
called. UnregisterThreadOnExit unregister_thread_on_exit;
    // Use DDS API
}
```

8.419.2 Constructor & Destructor Documentation

8.419.2.1 ~UnregisterThreadOnExit()

```
rti::core::UnregisterThreadOnExit::~UnregisterThreadOnExit ( )  [inline]
```

Calls **rti::core::unregister_thread** (p. 437).

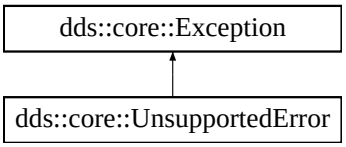
References **rti::core::unregister_thread()**.

8.420 dds::core::UnsupportedError Class Reference

Indicates that the application used an unsupported operation.

```
#include <Exception.hpp>
```

Inheritance diagram for dds::core::UnsupportedError:



Public Member Functions

- virtual const char * **what** () const throw ()
  *Access the message contained in this **UnsupportedError** (p. 2590) exception.*

8.420.1 Detailed Description

Indicates that the application used an unsupported operation.

Inherits also from `std::logic_error`

Only unsupported operations can throw this exception.

8.420.2 Member Function Documentation

8.420.2.1 what()

```
virtual const char * dds::core::UnsupportedError::what ( ) const throw ( )   [virtual]
```

Access the message contained in this **UnsupportedError** (p. 2590) exception.

Returns

The message.

Implements **dds::core::Exception** (p. 1547).

8.421 dds::core::policy::UserData Class Reference

Attaches a buffer of opaque data that is distributed by **Built-in Topics** (p. 46) during discovery.

```
#include <dds/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **UserData** ()
  Creates an instance with an empty sequence of bytes.
- **UserData** (const **dds::core::ByteSeq** &seq)
  Creates an instance with a sequence of bytes.
- **UserData** (const uint8_t *value_begin, const uint8_t *value_end)
  Creates an instance with a sequence of bytes.
- template<typename OctetIter >
  UserData & **value** (OctetIter the_begin, OctetIter the_end)
  Sets the byte sequence.
- const **dds::core::ByteSeq** **value** () const
  Gets the user data.
- **dds::core::ByteSeq** & **value** (**dds::core::ByteSeq** &dest) const
  Gets the user data.
- const uint8_t * **begin** () const
  *Gets the beginning of the range of bytes in this **UserData** (p. 2591).*
- const uint8_t * **end** () const
  *Gets the end of the range of bytes in this **UserData** (p. 2591).*

8.421.1 Detailed Description

Attaches a buffer of opaque data that is distributed by **Built-in Topics** (p. 46) during discovery.

Entity:

dds::domain::DomainParticipant (p. 1326), **dds::sub::DataReader** (p. 976), **dds::pub::DataWriter** (p. 1138)

Properties:

RxO (p. 514) = NO;
Changeable (p. 514) = **YES** (p. 514)

See also

dds::sub::builtin_subscriber (p. 671)

8.421.2 Usage

The purpose of this QoS is to allow the application to attach additional information to the created **dds::core::Entity** (p. 1520) objects, so that when a remote application discovers their existence, it can access that information and use it for its own purposes. This information is not used by RTI Connext.

One possible use of this QoS is to attach security credentials or some other information that can be used by the remote application to authenticate the source.

In combination with operations such as **dds::domain::ignore** (p. 635), **dds::pub::ignore** (p. 647), **dds::sub::ignore** (p. 667), and **dds::topic::ignore()** (p. 693), this QoS policy can assist an application to define and enforce its own security policies.

The use of this QoS is not limited to security; it offers a simple, yet flexible extensibility mechanism.

Important: RTI Connext stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Connext with the maximum size of the data that will be stored in policies of this type. This size is configured with **rti::core::policy::DomainParticipantResourceLimits::participant_user_data_max_length** (p. 1414), **rti::core::policy::DomainParticipantResourceLimits::writer_user_data_max_length** (p. 1416), and **rti↩ ::core::policy::DomainParticipantResourceLimits::reader_user_data_max_length** (p. 1416).

8.421.3 Constructor & Destructor Documentation

8.421.3.1 UserData() [1/3]

```
dds::core::policy::UserData::UserData ( )  [inline]
```

Creates an instance with an empty sequence of bytes.

8.421.3.2 UserData() [2/3]

```
dds::core::policy::UserData::UserData (
            const  dds::core::ByteSeq & seq )  [inline], [explicit]
```

Creates an instance with a sequence of bytes.

Parameters

seq	A vector containing the bytes to create this **UserData** (p. 2591)

8.421.3.3 UserData() [3/3]

```
dds::core::policy::UserData::UserData (
            const uint8_t * value_begin,
            const uint8_t * value_end )  [inline]
```

Creates an instance with a sequence of bytes.

Parameters

value_begin	Beginning of a range of bytes
value_end	End of the reange

8.421.4 Member Function Documentation

8.421.4.1 value() [1/3]

```
template<typename OctetIter >
UserData & dds::core::policy::UserData::value (
            OctetIter the_begin,
            OctetIter the_end )  [inline]
```

Sets the byte sequence.

Template Parameters

OctetIter	An input iterator of bytes (convertible to uint8_t).

Parameters

the_begin	Beginning of the range
the_end	End of the range

8.421.4.2 value() [2/3]

```
const  dds::core::ByteSeq dds::core::policy::UserData::value ( ) const  [inline]
```

Gets the user data.

Returns

the sequence of bytes representing the user data

8.421.4.3 value() [3/3]

```
dds::core::ByteSeq & dds::core::policy::UserData::value (
            dds::core::ByteSeq & dest ) const  [inline]
```

Gets the user data.

Parameters

dest	The vector where the bytes will be copied

Returns

A reference to `dest`

8.421.4.4 begin()

```
const uint8_t * dds::core::policy::UserData::begin ( ) const  [inline]
```

Gets the beginning of the range of bytes in this **UserData** (p. 2591).

8.421.4.5 end()

```
const uint8_t * dds::core::policy::UserData::end ( ) const  [inline]
```

Gets the end of the range of bytes in this **UserData** (p. 2591).

8.422 rti::sub::ValidLoanedSamples< T > Class Template Reference

<<***extension***>> *(p. 354)* <<***C++11***>> *(p. 353)* <<***move-only-type***>> *(p. 353)* Provides access to only those samples that contain valid data

`#include <LoanedSamplesImpl.hpp>`

Public Types

- typedef **ValidSampleIterator**< T > **iterator**
  *The iterator type that **ValidLoanedSamples** (p. 2595) provides.*
- typedef **ValidSampleIterator**< T > **const_iterator**
  *The const-iterator type that **ValidLoanedSamples** (p. 2595) provides.*
- typedef **ValidSampleIterator**< T >::value_type **value_type**
  This collection's value type, LoanedSample<T>

Public Member Functions

- **ValidLoanedSamples** (LoanedSamples< T > &&loaned_samples)
  Constructor from an rvalue LoanedSamples.
- **ValidLoanedSamples** (**ValidLoanedSamples**< T > &&other)
  Move constructor.
- **iterator begin** ()
  Returns a forward iterator to the first valid-data sample.
- **const_iterator begin** () const
  Returns a forward const iterator to the first valid-data sample.
- **iterator end** ()
  Returns a forward iterator that indicates the end of the collection.
- **const_iterator end** () const
  Returns a forward const iterator that indicates the end of the collection.

Related Functions

(Note that these are not member functions.)

- template<typename T >
  ValidLoanedSamples< T > **::iterator begin** (**ValidLoanedSamples**< T > &ls)
- template<typename T >
  ValidLoanedSamples< T > **::const_iterator begin** (const **ValidLoanedSamples**< T > &ls)
- template<typename T >
  ValidLoanedSamples< T > **::iterator end** (**ValidLoanedSamples**< T > &ls)
- template<typename T >
  ValidLoanedSamples< T > **::const_iterator end** (const **ValidLoanedSamples**< T > &ls)
- template<typename T >
  void **swap** (**ValidLoanedSamples**< T > &ls1, **ValidLoanedSamples**< T > &ls2) throw()

8.422.1 Detailed Description

template<typename T>
class rti::sub::ValidLoanedSamples< T >

*<<**extension**>> (p. 354) <<**C++11**>> (p. 353) <<**move-only-type**>> (p. 353)* Provides access to only those samples that contain valid data

Template Parameters

T	The topic-type. It has to match the type of the DataReader.

An instance of **ValidLoanedSamples** (p. 2595) is a wrapper that takes an existing LoanedSamples and provides forward iterators that only access samples with valid data.

The typical way to use this collection is through the function **rti::sub::valid_data** (p. 766), which transforms a **dds::sub::LoanedSamples** (p. 1663) into a **ValidLoanedSamples** (p. 2595).

For more information and a code example, see **rti::sub::valid_data(LoanedSamples<T>&&)** (p. 766)

See also

Reading data samples (p. 318)

8.422.2 Member Typedef Documentation

8.422.2.1 iterator

```
template<typename T >
typedef  ValidSampleIterator<T>  rti::sub::ValidLoanedSamples< T >::iterator
```

The iterator type that **ValidLoanedSamples** (p. 2595) provides.

8.422.2.2 const_iterator

```
template<typename T >
typedef  ValidSampleIterator<T>  rti::sub::ValidLoanedSamples< T >::const_iterator
```

The const-iterator type that **ValidLoanedSamples** (p. 2595) provides.

8.422.2.3 value_type

```
template<typename T >
typedef  ValidSampleIterator<T>::value_type  rti::sub::ValidLoanedSamples< T >::value_type
```

This collection's value type, LoanedSample<T>

8.422.3 Constructor & Destructor Documentation

8.422.3.1 ValidLoanedSamples() [1/2]

```
template<typename T >
rti::sub::ValidLoanedSamples< T >::ValidLoanedSamples (
            LoanedSamples< T > && loaned_samples )  [inline]
```

Constructor from an rvalue LoanedSamples.

The only way to create a **ValidLoanedSamples** (p. 2595) is through moving an existing LoanedSamples.

Postcondition

loaned_samples is invalid should not be used; this **ValidLoanedSamples** (p. 2595) now owns it.

Note

This constructor should not be used directly, the function **rti::sub::valid_data(LoanedSamples<T>&&)** (p. 766) is more convenient since it can deduce the template type.

8.422.3.2 ValidLoanedSamples() [2/2]

```
template<typename T >
rti::sub::ValidLoanedSamples< T >::ValidLoanedSamples (
             ValidLoanedSamples< T > && other )  [inline]
```

Move constructor.

A **ValidLoanedSamples** (p. 2595), just like LoanedSamples can be moved but not copied.

8.422.4 Member Function Documentation

8.422.4.1 begin() [1/2]

```
template<typename T >
iterator rti::sub::ValidLoanedSamples< T >::begin ( ) [inline]
```

Returns a forward iterator to the first valid-data sample.

Referenced by **rti::sub::ValidLoanedSamples< T >::begin()**.

8.422.4.2 begin() [2/2]

```
template<typename T >
const_iterator rti::sub::ValidLoanedSamples< T >::begin ( ) const [inline]
```

Returns a forward const iterator to the first valid-data sample.

8.422.4.3 end() [1/2]

```
template<typename T >
iterator rti::sub::ValidLoanedSamples< T >::end ( ) [inline]
```

Returns a forward iterator that indicates the end of the collection.

Referenced by **rti::sub::ValidLoanedSamples< T >::end()**.

8.422.4.4 end() [2/2]

```
template<typename T >
const_iterator rti::sub::ValidLoanedSamples< T >::end ( ) const [inline]
```

Returns a forward const iterator that indicates the end of the collection.

8.422.5 Friends And Related Function Documentation

8.422.5.1 begin() [1/2]

```
template<typename T >
ValidLoanedSamples< T > ::iterator begin (
            ValidLoanedSamples< T > & ls )  [related]
```

See also

ValidLoanedSamples::begin() (p. 2597)

References **rti::sub::ValidLoanedSamples< T >::begin()**.

8.422.5.2 begin() [2/2]

```
template<typename T >
ValidLoanedSamples< T > ::const_iterator begin (
          const  ValidLoanedSamples< T > & ls )  [related]
```

See also

ValidLoanedSamples::begin() (p. 2597)

References **rti::sub::ValidLoanedSamples< T >::begin()**.

8.422.5.3 end() [1/2]

```
template<typename T >
ValidLoanedSamples< T > ::iterator end (
            ValidLoanedSamples< T > & ls )  [related]
```

See also

ValidLoanedSamples::end() (p. 2598)

References **rti::sub::ValidLoanedSamples< T >::end()**.

8.422.5.4 end() [2/2]

```
template<typename T >
ValidLoanedSamples< T > ::const_iterator end (
            const  ValidLoanedSamples< T > & ls )  [related]
```

See also

ValidLoanedSamples::end() (p. 2598)

References **rti::sub::ValidLoanedSamples< T >::end()**.

8.422.5.5 swap()

```
template<typename T >
void swap (
             ValidLoanedSamples< T > & ls1,
             ValidLoanedSamples< T > & ls2 ) throw( )   [related]
```

See also

ValidLoanedSamples::swap() (p. 2600)

8.423 rti::sub::ValidSampleIterator< T > Class Template Reference

A forward iterator adapter that skips invalid samples.

`#include <SampleIterator.hpp>`

8.423.1 Detailed Description

template<typename T>
class rti::sub::ValidSampleIterator< T >

A forward iterator adapter that skips invalid samples.

Instead of instantiating this class directly, use **valid_data()** (p. 766).

See also

rti::sub::valid_data(const SampleIterator<T> &) (p. 767)

rti::sub::valid_data(LoanedSamples<T>&&) (p. 766)

8.424 dds::core::Value< D > Class Template Reference

`#include <Value.hpp>`

Public Member Functions

- const D * **operator->** () const OMG_NOEXCEPT
- D * **operator->** () OMG_NOEXCEPT
- const D & **delegate** () const OMG_NOEXCEPT
- D & **delegate** () OMG_NOEXCEPT
- const D & **extensions** () const OMG_NOEXCEPT
- D & **extensions** () OMG_NOEXCEPT
- **operator D&** () OMG_NOEXCEPT
- **operator const D &** () const OMG_NOEXCEPT

8.424.1 Detailed Description

template<typename D>
class dds::core::Value< D >

This class provides the basic behavior for **Value** (p. 2601) types.

8.424.2 Member Function Documentation

8.424.2.1 operator->() [1/2]

```
template<typename D >
const D *  dds::core::Value< D >::operator-> ( ) const  [inline]
```

Return the delegate.

8.424.2.2 operator->() [2/2]

```
template<typename D >
D *  dds::core::Value< D >::operator-> ( )  [inline]
```

Return the delegate.

8.424.2.3 delegate() [1/2]

```
template<typename D >
const D &  dds::core::Value< D >::delegate ( ) const  [inline]
```

Return the delegate.

Referenced by **dds::core::StringTopicType::data()**, **dds::topic::PublicationBuiltinTopicData::data_tag()**, **dds::topic::TopicBuiltinTopicData::destination_order()**, **dds::topic::TopicBuiltinTopicData::durability_service()**, **dds::topic::PublicationBuiltinTopicData::group_data()**, **dds::topic::TopicBuiltinTopicData::history()**, **dds::core::KeyedStringTopicType::key()**, **dds::topic::TopicBuiltinTopicData::key()**, **dds::core::KeyedBytesTopicType::key()**, **dds::topic::TopicBuiltinTopicData::latency_budget()**, **dds::core::BytesTopicType::length()**, **dds::core::KeyedBytesTopicType::length()**, **dds::topic::TopicBuiltinTopicData::lifespan()**, **dds::topic::TopicBuiltinTopicData::liveliness()**, **dds::topic::TopicBuiltinTopicData::name()**, **dds::core::KeyedBytesTopicType::operator std::vector< uint8_t >()**, **dds::core::BytesTopicType::operator[]()**, **dds::core::KeyedBytesTopicType::operator[]()**, **dds::topic::TopicBuiltinTopicData::ownership()**, **dds::topic::PublicationBuiltinTopicData::partition()**, **dds::topic::TopicBuiltinTopicData::reliability()**, **dds::topic::TopicBuiltinTopicData::representation()**, **dds::topic::PublicationBuiltinTopicData::representation()**, **dds::topic::TopicBuiltinTopicData::resource_limits()**, **dds::pub::SuspendedPublication::resume()**, **dds::topic::TopicBuiltinTopicData::topic_data()**, **dds::topic::PublicationBuiltinTopicData::topic_data()**, **dds::topic::TopicBuiltinTopicData::transport_priority()**, **dds::topic::TopicBuiltinTopicData::type_name()**, **dds::core::KeyedStringTopicType::value()**, and **dds::core::KeyedBytesTopicType::value()**.

8.424.2.4 delegate() [2/2]

```
template<typename D >
D &  dds::core::Value< D >::delegate ( )  [inline]
```

Return the delegate.

8.424.2.5 extensions() [1/2]

```
template<typename D >
const D &  dds::core::Value< D >::extensions ( ) const  [inline]
```

Provides access to the extensions (just like **delegate()** (p. 2602))

8.424.2.6 extensions() [2/2]

```
template<typename D >
D &  dds::core::Value< D >::extensions ( )  [inline]
```

Provides access to the extensions (just like delegate)

8.424.2.7 operator D&()

```
template<typename D >
dds::core::Value< D >::operator D& ( )  [inline]
```

Return the delegate.

8.424.2.8 operator const D &()

```
template<typename D >
dds::core::Value< D >::operator const D & ( ) const  [inline]
```

Return the delegate.

8.425 dds::core::vector< T > Class Template Reference

<<***value-type***>> *(*p. *350)* A vector convertible to std::vector and with similar functionality

```
#include <dds/core/vector.hpp>
```

Public Member Functions

- **vector** ()
  Creates an empty vector.
- **vector** (size_type initial_size)
  Creates a vector with a number of default-constructed elements.
- **vector** (size_type initial_size, const T &value)
  Creates a vector with a number of copies of a value.
- **vector** (const **vector** &other)
  Creates a copy.
- **vector** (const std::vector< T > &other)
  Implicit construction from std::vector.
- **vector** (**vector** &&other) OMG_NOEXCEPT
  Creates a vector by moving an existing one.
- **operator std::vector**< **T** > () const
  Provides a conversion to std::vector.
- size_type **size** () const
  Get the current size.
- size_type **capacity** () const
  Get the current capacity.
- void **resize** (size_type new_size)
  Set the size to new_size.
- void **resize** (size_type new_size, const T &value)
  Set the size to new_size, copying value if new elements are added.
- void **clear** ()

Resize to zero.

- void **reserve** (size_type new_capacity)

  Reserve a new capacity without creating any elements or changing the size.
- reference **at** (size_type i)

  Get an element by reference and check bounds.
- const_reference **at** (size_type i) const

  Get an element by const-reference and check bounds.
- reference **operator[ ]** (size_type i)

  Get an element by reference.
- const_reference **operator[ ]** (size_type i) const

  Get an element by const-reference.
- **vector** & **operator=** (const **vector** &other)

  Assign another vector to this one.
- **vector** & **operator=** (**vector** &&other) OMG_NOEXCEPT

  Move-assign another vector to this one.
- bool **operator==** (const **vector** &other) const

  Returns if the elements of two vectors are equal.
- bool **operator!=** (const **vector** &other) const

  Returns if two vectors are different.
- iterator **begin** ()

  Iterator to the first element.
- const_iterator **begin** () const

  Const iterator to first element.
- iterator **end** ()

  Iterator to last element plus one.
- const_iterator **end** () const

  Const iterator to last element plus one.

Friends

- void **swap** (**vector** &left, **vector** &right) OMG_NOEXCEPT

  Efficiently swaps the contents of two vectors.

Related Functions

(Note that these are not member functions.)

- template<typename T , size_t N>
  std::ostream & **operator**<< (std::ostream &out, const bounded_sequence< T, N > &v)

  Print a vector applying << to all of its elements.
- template<typename T >
  std::ostream & **operator**<< (std::ostream &out, const **vector**< T > &v)

  Print a vector applying << to all of its elements.

8.425.1 Detailed Description

template<typename T>
class dds::core::vector< T >

<<***value-type***>> *(p. 350)* A vector convertible to std::vector and with similar functionality

In many cases, for performance reasons and other implementation requirements, the RTI Connext API uses **dds::core::vector** (p. 2603) instead of std::vector. The most significant case is the C++ types that rtiddsgen generates from **IDL** (p. 606).

A **dds::core::vector** (p. 2603) provides a subset of the functionality of a `std::vector`, including iterators, resizing and indexed access. It also provides automatic conversion to and from std::vector, making code like the following possible:

```
// IDL-generated type
class Foo {
public:
    // ...
    const dds::core::vector<int32_t>& my_ints() const;
};
// Application
Foo sample = ...;
dds::vector<int32_t> my_std_vector = sample.my_ints(); // conversion to std::vector
// ...
sample.my_ints() = my_std_vector; // conversion from std::vector
```

8.425.2 Constructor & Destructor Documentation

8.425.2.1 vector() [1/6]

```
template<typename T >
dds::core::vector< T >::vector ( )  [inline]
```

Creates an empty vector.

8.425.2.2 vector() [2/6]

```
template<typename T >
dds::core::vector< T >::vector (
            size_type initial_size )  [inline], [explicit]
```

Creates a vector with a number of default-constructed elements.

8.425.2.3 vector() [3/6]

```
template<typename T >
dds::core::vector< T >::vector (
            size_type initial_size,
            const T & value )  [inline]
```

Creates a vector with a number of copies of a value.

8.425.2.4 vector() [4/6]

```
template<typename T >
dds::core::vector< T >::vector (
            const  vector< T > & other )  [inline]
```

Creates a copy.

8.425.2.5 vector() [5/6]

```
template<typename T >
dds::core::vector< T >::vector (
            const std::vector< T > & other )  [inline]
```

Implicit construction from std::vector.

Note: this operation makes a copy of all the elements in `other`

8.425.2.6 vector() [6/6]

```
template<typename T >
dds::core::vector< T >::vector (
             vector< T > && other )  [inline]
```

Creates a vector by moving an existing one.

8.425.3 Member Function Documentation

8.425.3.1 operator std::vector< T >()

```
template<typename T >
dds::core::vector< T >::operator std::vector< T > ( ) const  [inline]
```

Provides a conversion to std::vector.

Note: this operation makes a copy of all the elements in this vector

8.425.3.2 size()

```
template<typename T >
size_type  dds::core::vector< T >::size ( ) const  [inline]
```

Get the current size.

8.425.3.3 capacity()

```
template<typename T >
size_type  dds::core::vector< T >::capacity ( ) const  [inline]
```

Get the current capacity.

8.425.3.4 resize() [1/2]

```
template<typename T >
void  dds::core::vector< T >::resize (
            size_type new_size )  [inline]
```

Set the size to new_size.

8.425.3.5 resize() [2/2]

```
template<typename T >
void  dds::core::vector< T >::resize (
            size_type new_size,
            const T & value )  [inline]
```

Set the size to new_size, copying value if new elements are added.

8.425.3.6 clear()

```
template<typename T >
void  dds::core::vector< T >::clear ( )  [inline]
```

Resize to zero.

8.425.3.7 reserve()

```
template<typename T >
void  dds::core::vector< T >::reserve (
            size_type new_capacity )  [inline]
```

Reserve a new capacity without creating any elements or changing the size.

8.425.3.8 at() [1/2]

```
template<typename T >
reference  dds::core::vector< T >::at (
            size_type i )  [inline]
```

Get an element by reference and check bounds.

Exceptions

std::out_of_range	if `i` is out of bounds

8.425.3.9 at() [2/2]

```
template<typename T >
const_reference  dds::core::vector< T >::at (
            size_type i ) const  [inline]
```

Get an element by const-reference and check bounds.

Exceptions

std::out_of_range	if `i` is out of bounds

8.425.3.10 operator[]() [1/2]

```
template<typename T >
reference  dds::core::vector< T >::operator[] (
             size_type i )  [inline]
```

Get an element by reference.

8.425.3.11 operator[]() [2/2]

```
template<typename T >
const_reference  dds::core::vector< T >::operator[] (
             size_type i ) const  [inline]
```

Get an element by const-reference.

8.425.3.12 operator=() [1/2]

```
template<typename T >
vector &  dds::core::vector< T >::operator= (
             const  vector< T > & other )  [inline]
```

Assign another vector to this one.

8.425.3.13 operator=() [2/2]

```
template<typename T >
vector &  dds::core::vector< T >::operator= (
              vector< T > && other )  [inline]
```

Move-assign another vector to this one.

8.425.3.14 operator==()

```
template<typename T >
bool  dds::core::vector< T >::operator== (
            const  vector< T > & other ) const  [inline]
```

Returns if the elements of two vectors are equal.

8.425.3.15 operator"!=()

```
template<typename T >
bool  dds::core::vector< T >::operator!= (
            const  vector< T > & other ) const  [inline]
```

Returns if two vectors are different.

8.425.3.16 begin() [1/2]

```
template<typename T >
iterator  dds::core::vector< T >::begin ( )  [inline]
```

Iterator to the first element.

8.425.3.17 begin() [2/2]

```
template<typename T >
const_iterator  dds::core::vector< T >::begin ( ) const  [inline]
```

Const iterator to first element.

8.425.3.18 end() [1/2]

```
template<typename T >
iterator  dds::core::vector< T >::end ( )  [inline]
```

Iterator to last element plus one.

8.425.3.19 end() [2/2]

```
template<typename T >
const_iterator  dds::core::vector< T >::end ( ) const  [inline]
```

Const iterator to last element plus one.

8.425.4 Friends And Related Function Documentation

8.425.4.1 swap

```
template<typename T >
void swap (
              vector< T > & left,
              vector< T > & right )  [friend]
```

Efficiently swaps the contents of two vectors.

8.425.4.2 operator<<() [1/2]

```
template<typename T , size_t N>
std::ostream & operator<< (
             std::ostream & out,
             const bounded_sequence< T, N > & v )  [related]
```

Print a vector applying << to all of its elements.

8.425.4.3 operator<<() [2/2]

```
template<typename T >
std::ostream & operator<< (
             std::ostream & out,
             const  vector< T > & v )  [related]
```

Print a vector applying << to all of its elements.

8.426 rti::core::VendorId Class Reference

<<***extension***>> *(*p. *354)* Represents the vendor of the service implementing the RTPS protocol.

`#include <rti/core/VendorId.hpp>`

Public Member Functions

- **VendorId** ()

  *Creates the **unknown()** (*p. *2612) vendor id.*
- std::vector< uint8_t > **value** () const

  Provides access to the bytes that represent the vendor id.

Static Public Member Functions

- static **VendorId unknown** ()

  The ID that indicates that the vendor is unknown.

8.426.1 Detailed Description

<<***extension***>> *(*p. *354)* Represents the vendor of the service implementing the RTPS protocol.

8.426.2 Constructor & Destructor Documentation

8.426.2.1 VendorId()

`rti::core::VendorId::VendorId ( )  [inline]`

Creates the **unknown()** (p. 2612) vendor id.

8.426.3 Member Function Documentation

8.426.3.1 value()

`std::vector< uint8_t > rti::core::VendorId::value ( ) const  [inline]`

Provides access to the bytes that represent the vendor id.

8.426.3.2 unknown()

```
static VendorId rti::core::VendorId::unknown ( ) [inline], [static]
```

The ID that indicates that the vendor is unknown.

8.427 rti::config::Verbosity_def Struct Reference

The definition of the **dds::core::safe_enum** (p. 2265) Verbosity.

```
#include <rti/config/Logger.hpp>
```

Public Types

- enum **type** {
  silent ,
  exception ,
  warning ,
  status_local ,
  status_remote ,
  status_all ,
  SILENT = silent ,
  EXCEPTION = exception ,
  WARNING = warning ,
  STATUS_LOCAL = status_local ,
  STATUS_REMOTE = status_remote ,
  STATUS_ALL = status_all ,
  ERRORY = EXCEPTION }

  The underlying `enum` type.

8.427.1 Detailed Description

The definition of the **dds::core::safe_enum** (p. 2265) Verbosity.

8.427.2 Member Enumeration Documentation

8.427.2.1 type

```
enum rti::config::Verbosity_def::type
```

The underlying `enum` type.

Enumerator

silent	No further output will be logged.
exception	Only error and fatal error messages will be logged. An error indicates something wrong in the functioning of RTI Connext. The most common cause of errors is incorrect configuration.
warning	Both error and warning messages will be logged. A warning indicates that RTI Connext is taking an action that may or may not be what you intended. Some configuration information is also logged at this verbosity to aid in debugging.
status_local	Errors, warnings, and verbose information about the lifecycles of local RTI Connext objects will be logged.
status_remote	Errors, warnings, and verbose information about the lifecycles of remote RTI Connext objects will be logged.
status_all	Errors, warnings, verbose information about the lifecycles of local and remote RTI Connext objects, and periodic information about RTI Connext threads will be logged.

8.428 dds::sub::status::ViewState Class Reference

Indicates whether or not an instance is new.

`#include <dds/sub/status/DataState.hpp>`

Inherits std::bitset< OMG_DDS_STATE_BIT_COUNT >.

Public Types

- typedef std::bitset< OMG_DDS_STATE_BIT_COUNT > **MaskType**

  An std::bitset of ViewStates.

Public Member Functions

- **ViewState** (const **MaskType** &other)

  *Create an **ViewState** (p. 2614) from MaskType.*

Static Public Member Functions

- static const **ViewState** **new_view** ()

  *Creates a new_view **ViewState** (p. 2614) object.*
- static const **ViewState** **not_new_view** ()

  *Creates a not_new_view **ViewState** (p. 2614) object.*
- static const **ViewState** **any** ()

  *Creates a **ViewState** (p. 2614) object representing any view state.*

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator**<< (std::ostream &os, const **ViewState** &s)
  Prints a view state as a readable string.

8.428.1 Detailed Description

Indicates whether or not an instance is new.

For each instance (identified by the key), the middleware internally maintains a view state relative to each **dds::sub::DataReader** (p. 976). The view state can be either:

- **dds::sub::status::ViewState::new_view()** (p. 2616) indicates that either this is the first time that the **dds::sub::DataReader** (p. 976) has ever accessed samples of that instance, or else that the **dds::sub::DataReader** (p. 976) has accessed previous samples of the instance, but the instance has since been reborn (i.e. become not-alive and then alive again). These two cases are distinguished by examining the **dds::sub::GenerationCount::disposed()** (p. 1590) and the **dds::sub::GenerationCount::no_writers()** (p. 1590).

- **dds::sub::status::ViewState::not_new_view()** (p. 2616) indicates that the **dds::sub::DataReader** (p. 976) has already accessed samples of the same instance and that the instance has not been reborn since.

The view_state available in the **dds::sub::SampleInfo** (p. 2285) is a snapshot of the view state of the instance relative to the **dds::sub::DataReader** (p. 976) used to access the samples at the time the collection was obtained (i.e. at the time read or take was called). The view_state is therefore the same for all samples in the returned collection that refer to the same instance.

Once an instance has been detected as not having any "live" writers and all the samples associated with the instance are "taken" from the **dds::sub::DataReader** (p. 976), the middleware can reclaim all local resources regarding the instance. Future samples will be treated as "never seen."

8.428.2 Member Typedef Documentation

8.428.2.1 MaskType

```
typedef std::bitset<OMG_DDS_STATE_BIT_COUNT >  dds::sub::status::ViewState::MaskType
```

An std::bitset of ViewStates.

8.428.3 Constructor & Destructor Documentation

8.428.3.1 ViewState()

```
dds::sub::status::ViewState::ViewState (
            const  MaskType & other )  [inline]
```

Create an **ViewState** (p. 2614) from MaskType.

Parameters

other	The MaskType to create the **ViewState** (p. 2614) with

8.428.4 Member Function Documentation

8.428.4.1 new_view()

```
static const  ViewState dds::sub::status::ViewState::new_view ( )  [inline], [static]
```

Creates a new_view **ViewState** (p. 2614) object.

A new_view **ViewState** (p. 2614) indicates that the latest generation of the instance has not previously been accessed.

Referenced by **dds::sub::status::DataState::new_instance()**.

8.428.4.2 not_new_view()

```
static const  ViewState dds::sub::status::ViewState::not_new_view ( )  [inline], [static]
```

Creates a not_new_view **ViewState** (p. 2614) object.

Indicates that the latest generation of an instance has previously been accessed.

8.428.4.3 any()

```
static const  ViewState dds::sub::status::ViewState::any ( )  [inline], [static]
```

Creates a **ViewState** (p. 2614) object representing any view state.

This is equivalent to **new_view()** (p. 2616) | **not_new_view()** (p. 2616)

Referenced by **dds::sub::status::DataState::any()**, **dds::sub::status::DataState::any_data()**, and **dds::sub↩ ::status::DataState::new_data()**.

8.428.5 Friends And Related Function Documentation

8.428.5.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const  ViewState & s )  [related]
```

Prints a view state as a readable string.

8.429 dds::core::cond::WaitSet Class Reference

<<*reference-type*>> *(*p. *351)* Allows an application to wait until one or more of the attached **Condition** (p. 948) objects have a trigger_value of true or else until the timeout expires.

```
#include <dds/core/cond/WaitSet.hpp>
```

Public Types

- typedef std::vector< **dds::core::cond::Condition** > **ConditionSeq**

  A vector of Conditions.

Public Member Functions

- **WaitSet** ()

  *Creates a **WaitSet** (*p. *2617) with no conditions attached.*
- **WaitSetImpl** (const **rti::core::cond::WaitSetProperty** & **property**)

  <<***extension***>> *(*p. *354) Creates a **WaitSet** (*p. *2617) with a WaitSetProperty*
- const **ConditionSeq** **wait** (const **dds::core::Duration** &timeout)

  Allows an application thread to wait for the occurrence of certain conditions.
- const **ConditionSeq** **wait** ()

  Allows an application thread to wait for the occurrence of certain conditions.
- **ConditionSeq** & **wait** (**ConditionSeq** &triggered, const **dds::core::Duration** &timeout)

  Allows an application thread to wait for the occurrence of certain conditions.
- **ConditionSeq** & **wait** (**ConditionSeq** &triggered)

  Allows an application thread to wait for the occurrence of certain conditions.
- void **dispatch** (const **dds::core::Duration** &timeout)

  Waits for at least one of the attached conditions to trigger and then dispatches the events.
- void **dispatch** ()

  Waits for at least one of the attached conditions to trigger and then dispatches the events.
- **WaitSet** & **operator+=** (**dds::core::cond::Condition** cond)

  *Same as **attach_condition()** (*p. *2624)*
- **WaitSet** & **operator-=** (**dds::core::cond::Condition** cond)

  *Same as **detach_condition()** (*p. *2625).*
- **WaitSet** & **attach_condition** (**dds::core::cond::Condition** cond)

  *Attaches a **Condition** (*p. *948) to the **WaitSet** (*p. *2617).*
- bool **detach_condition** (**dds::core::cond::Condition** cond)

*Detaches a **Condition** (p. 948) from the **WaitSet** (p. 2617).*

- **ConditionSeq conditions** () const

  Retrieves the list of attached conditions.

- **ConditionSeq** & **conditions** (**ConditionSeq** &conds) const

  Retrieves the list of attached conditions.

- bool **wait_one** (const **dds::core::Duration** &timeout)

  *<<**extension**>> (p. 354) Simpler **wait()** (p. 2622) when only one condition is attached*

- void **detach_all** ()

  *<<**extension**>> (p. 354) Detaches all the conditions.*

- **rti::core::cond::WaitSetProperty property** () const

  *<<**extension**>> (p. 354) Gets the WaitSetProperty*

- void **property** (const **rti::core::cond::WaitSetProperty** &property)

  *<<**extension**>> (p. 354) Sets the property to configure the **WaitSet** (p. 2617) to return after one or more trigger events have occurred.*

8.429.1 Detailed Description

*<<**reference-type**>> (p. 351)* Allows an application to wait until one or more of the attached **Condition** (p. 948) objects have a trigger_value of true or else until the timeout expires.

Note

A **WaitSet** (p. 2617) provides all the functions of a *<<**reference-type**>> (p. 351)* except **close()** (p. 1024) and retain().

8.429.2 Usage

dds::core::cond::Condition (p. 948) (s) (in conjunction with wait-sets) provide an alternative mechanism to allow the middleware to communicate communication status changes (including arrival of data) to the application.

This mechanism is wait-based. Its general use pattern is as follows:

- The application indicates which relevant information it wants to get by creating **dds::core::cond::Condition** (p. 948) objects (**dds::core::cond::StatusCondition** (p. 2372), **dds::sub::cond::ReadCondition** (p. 2133) or **dds::sub::cond::QueryCondition** (p. 2059)) and attaching them to a **dds::core::cond::WaitSet** (p. 2617).

- It then waits on that **dds::core::cond::WaitSet** (p. 2617) until the `trigger_value` of one or several **dds::core::cond::Condition** (p. 948) objects become true.

- It then uses the result of the wait (i.e., active_conditions, the list of **dds::core::cond::Condition** (p. 948) objects with `trigger_value` == true) to actually get the information:

  – by calling **dds::core::Entity::status_changes()** (p. 1525) and then `get_<communication_status>()` on the relevant **dds::core::Entity** (p. 1520), if the condition is a **dds::core::cond::StatusCondition** (p. 2372) and the status changes, refer to plain communication status;

- by calling **dds::core::Entity::status_changes()** (p. 1525) and then **dds::sub::find** (p. 672) on the relevant **dds::sub::Subscriber** (p. 2410) (and then **dds::sub::DataReader::read()** (p. 996) or **dds::sub::Data↩ Reader::take** (p. 997) on the returned **dds::sub::DataReader** (p. 976) objects), if the condition is a **dds↩ ::core::cond::StatusCondition** (p. 2372) and the status changes refers to **dds::core::status::Status↩ Mask::data_on_readers()** (p. 2382);
- by calling **dds::core::Entity::status_changes()** (p. 1525) and then **dds::sub::DataReader::read()** (p. 996) or **dds::sub::DataReader::take** (p. 997) on the relevant **dds::sub::DataReader** (p. 976), if the condition is a **dds::core::cond::StatusCondition** (p. 2372) and the status changes refers to **dds::core::status::↩ StatusMask::data_available()** (p. 2383);
- by calling directly dds::sub::DataReader::read_w_condition or dds::sub::DataReader::take_w_condition on a **dds::sub::DataReader** (p. 976) with the **dds::core::cond::Condition** (p. 948) as a parameter if it is a **dds::sub::cond::ReadCondition** (p. 2133) or a **dds::sub::cond::QueryCondition** (p. 2059).

Usually the first step is done in an initialization phase, while the others are put in the application main loop.

As there is no extra information passed from the middleware to the application when a wait returns (only the list of triggered **dds::core::cond::Condition** (p. 948) objects), **dds::core::cond::Condition** (p. 948) objects are meant to embed all that is needed to react properly when enabled. In particular, **dds::core::Entity** (p. 1520)-related conditions are related to exactly one **dds::core::Entity** (p. 1520) and cannot be shared.

The blocking behavior of the **dds::core::cond::WaitSet** (p. 2617) is illustrated below.

The result of a **dds::core::cond::WaitSet::wait** (p. 2621) operation depends on the state of the **dds::core::cond::↩ WaitSet** (p. 2617), which in turn depends on whether at least one attached **dds::core::cond::Condition** (p. 948) has a `trigger_value` of true. If the wait operation is called on **dds::core::cond::WaitSet** (p. 2617) with state BLOCKED, it will block the calling thread. If wait is called on a **dds::core::cond::WaitSet** (p. 2617) with state UNBLOCKED, it will return immediately. In addition, when the **dds::core::cond::WaitSet** (p. 2617) transitions from BLOCKED to UNBLOCKED it wakes up any threads that had called wait on it.

A key aspect of the **Condition** (p. 948) and **WaitSet** (p. 2617) mechanism is the setting of the `trigger_value` of each **dds::core::cond::Condition** (p. 948).

The **dds::core::cond::WaitSet** (p. 2617) cannot be used after calling DomainParticipantFactory::finalize_instance.

8.429.3 Trigger State of a dds::core::cond::StatusCondition

The `trigger_value` of a **dds::core::cond::StatusCondition** (p. 2372) is the boolean OR of the ChangedStatus↩ Flag of all the communication statuses (see **Status Kinds** (p. 429)) to which it is sensitive. That is, `trigger_value` == false only if all the values of the ChangedStatusFlags are false.

The sensitivity of the **dds::core::cond::StatusCondition** (p. 2372) to a particular communication status is controlled by the list of `enabled_statuses` set on the condition by means of the **dds::core::cond::StatusCondition::enabled↩ _statuses(const dds::core::status::StatusMask&)** (p. 2373) operation.

Once the `trigger_value` of a **StatusCondition** (p. 2372) becomes true, it remains true until the status that changed is reset. To reset a status, call the related get_*_status() operation. Or, in the case of the data available status, call **read()** (p. 1024), **take()** (p. 1024), or one of their variants. Therefore, if you are using a **dds::core::cond::Status↩ Condition** (p. 2372) on a **dds::core::cond::WaitSet** (p. 2617) to be notified of events, your thread will wake up when one of the statuses associated with the **StatusCondition** (p. 2372) becomes true. If you do not reset the status, the **StatusCondition** (p. 2372) `trigger_value` remains true and your **WaitSet** (p. 2617) will not block again; it will immediately wake up when you call **dds::core::cond::WaitSet::wait** (p. 2621).

8.429.4 Trigger State of a dds::sub::cond::ReadCondition

Similar to the **dds::core::cond::StatusCondition** (p. 2372), a **dds::sub::cond::ReadCondition** (p. 2133) also has a `trigger_value` that determines whether the attached **dds::core::cond::WaitSet** (p. 2617) is BLOCKED or UNBLOCKED. However, unlike the **dds::core::cond::StatusCondition** (p. 2372), the `trigger_value` of the **dds::sub::cond::ReadCondition** (p. 2133) is tied to the presence of *at least a sample* managed by RTI Connext with **dds::sub::status::SampleState** (p. 2316) and **dds::sub::status::ViewState** (p. 2614) matching those of the **dds::sub::cond::ReadCondition** (p. 2133). Furthermore, for the **dds::sub::cond::QueryCondition** (p. 2059) to have a `trigger_value` == true, the data associated with the sample must be such that the `query_expression` evaluates to true.

The fact that the trigger_value of a **dds::sub::cond::ReadCondition** (p. 2133) depends on the presence of samples on the associated **dds::sub::DataReader** (p. 976) implies that a single take operation can potentially change the `trigger_value` of several **dds::sub::cond::ReadCondition** (p. 2133) or **dds::sub::cond::QueryCondition** (p. 2059) conditions. For example, if all samples are taken, any **dds::sub::cond::ReadCondition** (p. 2133) and **dds::sub::cond::QueryCondition** (p. 2059) conditions associated with the **dds::sub::DataReader** (p. 976) that had their `trigger_value==TRUE` before will see the `trigger_value` change to FALSE. Note that this does not guarantee that **dds::core::cond::WaitSet** (p. 2617) objects that were separately attached to those conditions will not be woken up. Once we have `trigger_value==TRUE` on a condition, it may wake up the attached **dds::core::cond::WaitSet** (p. 2617), the condition transitioning to `trigger_value==FALSE` does not necessarily 'unwakeup' the **WaitSet** (p. 2617) as 'unwakening' may not be possible in general.

The consequence is that an application blocked on a **dds::core::cond::WaitSet** (p. 2617) may return from the wait with a list of conditions, some of which are not no longer 'active'. This is unavoidable if multiple threads are concurrently waiting on separate **dds::core::cond::WaitSet** (p. 2617) objects and taking data associated with the same **dds::sub::DataReader** (p. 976) entity.

To elaborate further, consider the following example: A **dds::sub::cond::ReadCondition** (p. 2133) that has a `sample_state_mask` = {**dds::sub::status::SampleState::not_read()** (p. 2318)} will have `trigger_value` of true whenever a new sample arrives and will transition to false as soon as all the newly-arrived samples are either read (so their sample state changes to READ) or taken (so they are no longer managed by RTI Connext). However if the same **dds::sub::cond::ReadCondition** (p. 2133) had a `sample_state_mask` = { **dds::sub::status::SampleState::read()** (p. 2318), **dds::sub::status::SampleState::not_read()** (p. 2318) }, then the `trigger_value` would only become false once all the newly-arrived samples are taken (it is not sufficient to read them as that would only change the sample state to READ), which overlaps the mask on the **dds::sub::cond::ReadCondition** (p. 2133).

8.429.5 Trigger State of a dds::core::cond::GuardCondition

The `trigger_value` of a **dds::core::cond::GuardCondition** (p. 1594) is completely controlled by the application via the operation **dds::core::cond::GuardCondition::trigger_value()** (p. 1595).

See also

Status Kinds (p. 429)

dds::core::cond::StatusCondition (p. 2372), **dds::core::cond::GuardCondition** (p. 1594)

Listener (p. 1637)

See also

Waitset Use Cases (p. 330)

Examples

Foo_subscriber.cxx.

8.429.6 Member Typedef Documentation

8.429.6.1 ConditionSeq

```
typedef std::vector< dds::core::cond::Condition> dds::core::cond::WaitSet::ConditionSeq
```

A vector of Conditions.

8.429.7 Constructor & Destructor Documentation

8.429.7.1 WaitSet()

```
dds::core::cond::WaitSet::WaitSet ( )  [inline]
```

Creates a **WaitSet** (p. 2617) with no conditions attached.

8.429.8 Member Function Documentation

8.429.8.1 WaitSetImpl()

```
dds::core::cond::WaitSet::WaitSetImpl (
            const  rti::core::cond::WaitSetProperty & property )
```

<<***extension***>> *(p. 354)* Creates a **WaitSet** (p. 2617) with a WaitSetProperty

8.429.8.2 wait() [1/4]

```
const  ConditionSeq dds::core::cond::WaitSet::wait (
            const  dds::core::Duration & timeout )  [inline]
```

Allows an application thread to wait for the occurrence of certain conditions.

If none of the conditions attached to the **dds::core::cond::WaitSet** (p. 2617) have a trigger_value of true, the wait operation will block, suspending the calling thread.

The result of the wait operation is the list of all the attached conditions that have a trigger_value of true (i.e., the conditions that unblocked the wait).

The wait operation takes a timeout argument that specifies the maximum duration for the wait. If this duration is exceeded and none of the attached **dds::core::cond::Condition** (p. 948) objects are true, wait will return an empty condition sequence.

Note: The resolution of the timeout period is constrained by the resolution of the system clock.

It is not allowable for more than one application thread to be waiting on the same **WaitSet** (p. 2617). If the wait operation is invoked on a **WaitSet** (p. 2617) that already has a thread blocking on it, the operation will return immediately with the value **dds::core::PreconditionNotMetError** (p. 1941).

Parameters

timeout	The maximum time to wait

Returns

A vector containing the active conditions or an empty vector if the operation times out.

Exceptions

dds::core::PreconditionNotMetError *(p. 1941)*	or one of the other **Standard Exceptions** (p. 428). Note however that this operation *does not* throw **dds::core::TimeoutError** (p. 2473).

8.429.8.3 wait() [2/4]

```
const  ConditionSeq dds::core::cond::WaitSet::wait ( )  [inline]
```

Allows an application thread to wait for the occurrence of certain conditions.

This function is equivalent to `wait(dds::core::Duration::infinite())`.

See also

wait(const dds::core::Duration&) (p. 2621)

8.429.8.4 wait() [3/4]

```
ConditionSeq & dds::core::cond::WaitSet::wait (
            ConditionSeq & triggered,
          const  dds::core::Duration & timeout )  [inline]
```

Allows an application thread to wait for the occurrence of certain conditions.

This function is equivalent to **`wait(const dds::core::Duration&)`** (p. 2621) but reuses an existing vector instead of returning a new one.

Parameters

triggered	A vector where the active conditions will be added. Any element already present in the vector will be removed. If the operation times out, this vector will be empty.
timeout	The wait timeout

Returns

A reference to `triggered` for convenience.

See also

wait(const dds::core::Duration&) (p. 2621)

8.429.8.5 wait() [4/4]

```
ConditionSeq & dds::core::cond::WaitSet::wait (
            ConditionSeq & triggered )  [inline]
```

Allows an application thread to wait for the occurrence of certain conditions.

This application is equivalent to wait(triggered, dds::core::Duration::infinite());

See also

wait(ConditionSeq&, const dds::core::Duration&) (p. 2622)

8.429.8.6 dispatch() [1/2]

```
void dds::core::cond::WaitSet::dispatch (
            const  dds::core::Duration & timeout )  [inline]
```

Waits for at least one of the attached conditions to trigger and then dispatches the events.

This is an alternative, simpler pattern to the use of **wait()** (p. 2622). Instead of receiving the list of active conditions, the application associates a handler (functor) to each conditions before attaching it to the **WaitSet** (p. 2617). Whenever this function wakes up it will call–in the current thread context–the handler of each triggered condition before returning.

Note that each condition can only have one handler, so if you want to attach it to multiple WaitSets you may need to use **wait()** (p. 2622) instead of **dispatch()** (p. 2624).

Parameters

timeout	The maximum time to wait. If that time elapses **dispatch()** (p. 2624) will return without having called any of the condition handlers.

MT Safety:

It is not thread-safe to reset or replace a condition's handler while **dispatch()** (p. 2624) is running. It should be detached first. Detaching a condition from a **WaitSet** (p. 2617) while **dispatch()** (p. 2624) is running is safe even if the condition is active, but whether the condition handler will be executed or not is undefined.

See also

WaitSet wait and dispatch examples (p. 330)

Examples

Foo_subscriber.cxx.

8.429.8.7 dispatch() [2/2]

```
void dds::core::cond::WaitSet::dispatch ( )  [inline]
```

Waits for at least one of the attached conditions to trigger and then dispatches the events.

This function is equivalent to dispatch(dds::core::Duration::infinite());

See also

dispatch(const dds::core::Duration&) (p. 2623)

8.429.8.8 operator+=()

```
WaitSet & dds::core::cond::WaitSet::operator+= (
            dds::core::cond::Condition cond )  [inline]
```

Same as **attach_condition()** (p. 2624)

8.429.8.9 operator-=()

```
WaitSet & dds::core::cond::WaitSet::operator-= (
            dds::core::cond::Condition cond )  [inline]
```

Same as **detach_condition()** (p. 2625).

8.429.8.10 attach_condition()

```
WaitSet & dds::core::cond::WaitSet::attach_condition (
            dds::core::cond::Condition cond )  [inline]
```

Attaches a **Condition** (p. 948) to the **WaitSet** (p. 2617).

It is possible to attach a **dds::core::cond::Condition** (p. 948) on a **dds::core::cond::WaitSet** (p. 2617) that is currently being waited upon (via the wait operation). In this case, if the **dds::core::cond::Condition** (p. 948) has a `trigger_value` of true, then attaching the condition will unblock the **dds::core::cond::WaitSet** (p. 2617).

If the same **dds::core::cond::Condition** (p. 948) is attached multiple times, only the first attachment has any effect. Subsequent attempts to attach the same **dds::core::cond::Condition** (p. 948) are silently ignored. As a result, the **dds::core::cond::Condition** (p. 948) is only considered attached once and must only be detached once.

Parameters

cond	<<***in***>> *(p. 355)* **Condition** (p. 948) to be attached.

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::OutOfResourcesError** (p. 1898).

8.429.8.11 detach_condition()

```
bool dds::core::cond::WaitSet::detach_condition (
            dds::core::cond::Condition cond )  [inline]
```

Detaches a **Condition** (p. 948) from the **WaitSet** (p. 2617).

It is possible to detach a **dds::core::cond::Condition** (p. 948) on a **dds::core::cond::WaitSet** (p. 2617) that is currently being waited upon (via the wait operation). If the **dds::core::cond::Condition** (p. 948) was not attached to the **dds::core::cond::WaitSet** (p. 2617), the operation will return **dds::core::InvalidArgumentError** (p. 1619).

If the same **dds::core::cond::Condition** (p. 948) had been previously attached multiple times, only a single detach is required; the **dds::core::cond::Condition** (p. 948) is considered attached only once.

Parameters

cond	<<***in***>> *(p. 355)* **Condition** (p. 948) to be detached.

Exceptions

One	of the **Standard Exceptions** (p. 428), or **dds::core::PreconditionNotMetError** (p. 1941).

8.429.8.12 conditions() [1/2]

```
ConditionSeq dds::core::cond::WaitSet::conditions ( ) const  [inline]
```

Retrieves the list of attached conditions.

Returns

the list of attached conditions.

8.429.8.13 conditions() [2/2]

```
ConditionSeq & dds::core::cond::WaitSet::conditions (
            ConditionSeq & conds ) const  [inline]
```

Retrieves the list of attached conditions.

Parameters

conds	A vector where the conditions will be aded

Returns

A reference to `conds`.

8.429.8.14 wait_one()

```
bool wait_one (
            const  dds::core::Duration & timeout )
```

<<***extension***>> *(*p. *354)* Simpler **wait()** (p. 2622) when only one condition is attached

Unlike the rest of **`wait()`** (p. 2622) methods, **`wait_one()`** (p. 2626) doesn't return the active conditions. It only returns whether any condition was triggered (`true`) or whether the wait timed out (`false`).

8.429.8.15 detach_all()

```
void detach_all ( )
```

<<***extension***>> *(*p. *354)* Detaches all the conditions.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.429.8.16 property() [1/2]

```
rti::core::cond::WaitSetProperty property ( ) const
```

<<***extension***>> *(*p. *354)* Gets the WaitSetProperty

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.429.8.17 property() [2/2]

```
void property (
            const  rti::core::cond::WaitSetProperty & property )
```

<<***extension***>> *(*p. *354)* Sets the property to configure the **WaitSet** (p. 2617) to return after one or more trigger events have occurred.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

8.430 rti::core::cond::WaitSetProperty Class Reference

<<***extension***>> *(*p. *354)* <<***value-type***>> *(*p. *350)* Specifies the **dds::core::cond::WaitSet** (p. 2617) behavior for multiple trigger events

```
#include <WaitSetImpl.hpp>
```

Inherits rti::core::NativeValueType< T, NATIVE_T, ADAPTER >.

Public Member Functions

- **WaitSetProperty** ()

  *Creates the default **WaitSetProperty** (p. 2627).*
- **WaitSetProperty** (int32_t the_max_event_count, const **dds::core::Duration** &the_max_event_delay)

  *Creates a **WaitSetProperty** (p. 2627) with the given parameters.*
- int32_t **max_event_count** () const

  Gets the max_event_count.
- **WaitSetProperty** & **max_event_count** (int32_t value)

  Sets the maximum number of trigger events to cause a WaitSet to wake up.
- **dds::core::Duration max_event_delay** () const

  Gets the max_event_delay.
- **WaitSetProperty** & **max_event_delay** (const **dds::core::Duration** &value)

  Sets the maximum delay from occurrence of first trigger event to cause a WaitSet to wake up.

8.430.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Specifies the **dds::core::cond::WaitSet** (p. 2617) behavior for multiple trigger events

In simple use, a **dds::core::cond::WaitSet** (p. 2617) returns when a single trigger event occurs on one of its attached **dds::core::cond::Condition** (p. 948) (s), or when the `timeout` maximum wait duration specified in the **dds::core::cond::WaitSet::wait** (p. 2621) call expires.

The **rti::core::cond::WaitSetProperty** (p. 2627) allows configuration of the waiting behavior of a **dds::core::cond::WaitSet** (p. 2617). If no conditions are true at the time of the call to wait, then the `max_event_count` parameter may be used to configure the WaitSet to wait for `max_event_count` trigger events to occur before returning, or to wait for up to `max_event_delay` time from the occurrence of the first trigger event before returning.

The `timeout` maximum wait duration specified in the **dds::core::cond::WaitSet::wait** (p. 2621) call continues to apply.

Entity:

dds::core::cond::WaitSet (p. 2617)

Properties:

RxO (p. 514) = N/A

Changeable (p. 514) = **YES** (p. 514)

8.430.2 Constructor & Destructor Documentation

8.430.2.1 WaitSetProperty() [1/2]

```
rti::core::cond::WaitSetProperty::WaitSetProperty ( )
```

Creates the default **WaitSetProperty** (p. 2627).

8.430.2.2 WaitSetProperty() [2/2]

```
rti::core::cond::WaitSetProperty::WaitSetProperty (
            int32_t the_max_event_count,
            const  dds::core::Duration & the_max_event_delay )
```

Creates a **WaitSetProperty** (p. 2627) with the given parameters.

8.430.3 Member Function Documentation

8.430.3.1 max_event_count() [1/2]

```
int32_t rti::core::cond::WaitSetProperty::max_event_count ( ) const
```

Gets the max_event_count.

See also

max_event_count(int32_t) (p. 2629)

8.430.3.2 max_event_count() [2/2]

```
WaitSetProperty & rti::core::cond::WaitSetProperty::max_event_count (
            int32_t value )
```

Sets the maximum number of trigger events to cause a WaitSet to wake up.

The **dds::core::cond::WaitSet** (p. 2617) will wait until up to `max_event_count` trigger events have occurred before returning. The **dds::core::cond::WaitSet** (p. 2617) may return earlier if either the `timeout` duration has expired, or `max_event_delay` has elapsed since the occurrence of the first trigger event. `max_event_count` may be used to "collect" multiple trigger events for processing at the same time.

[default] 1

[range] >= 1

8.430.3.3 max_event_delay() [1/2]

```
dds::core::Duration rti::core::cond::WaitSetProperty::max_event_delay ( ) const
```

Gets the max_event_delay.

See also

max_event_delay(const dds::core::Duration&) (p. 2630)

8.430.3.4 max_event_delay() [2/2]

```
WaitSetProperty & rti::core::cond::WaitSetProperty::max_event_delay (
            const  dds::core::Duration & value )
```

Sets the maximum delay from occurrence of first trigger event to cause a WaitSet to wake up.

The **dds::core::cond::WaitSet** (p. 2617) will return no later than `max_event_delay` after the first trigger event. `max_event_delay` may be used to establish a maximum latency for events reported by the **dds::core::cond::WaitSet** (p. 2617).

Note that **dds::core::TimeoutError** (p. 2473) is *not* returned if `max_event_delay` is exceeded. **dds::core::TimeoutError** (p. 2473) is returned only if the `timeout` duration expires before any trigger events occur.

[default] dds::core::Duration::infinite() (p. 1451)

8.431 dds::core::WeakReference< T > Class Template Reference

```
#include <WeakReference.hpp>
```

8.431.1 Detailed Description

template<typename T>
class dds::core::WeakReference< T >

Unsupported - **WeakReference** (p. 2630) is not currently supported.

8.432 rti::core::policy::WireProtocol Class Reference

<<***extension***>> *(p. 116)* Configures the write protocol of a **dds::domain::DomainParticipant** (p. 1326)

```
#include <rti/core/policy/CorePolicy.hpp>
```

Public Member Functions

- **WireProtocol** ()

  Creates the default policy.
- **WireProtocol** & **participant_id** (int32_t the_participant_id)

  <<***basic***>> (p. *104*) <<***extension***>> (p. *116*) *A value used to distinguish among different participants belonging to the same domain on the same host.*
- int32_t **participant_id** () const

  Getter (see the setter with the same name)
- **WireProtocol** & **rtps_host_id** (uint32_t the_rtps_host_id)

  <<***extension***>> (p. *116*) *The RTPS Host ID of the domain participant.*
- uint32_t **rtps_host_id** () const

  Getter (see the setter with the same name)
- **WireProtocol** & **rtps_app_id** (uint32_t the_rtps_app_id)

  <<***extension***>> (p. *116*) *The RTPS App ID of the domain participant.*
- uint32_t **rtps_app_id** () const

  Getter (see the setter with the same name)
- **WireProtocol** & **rtps_instance_id** (uint32_t the_rtps_instance_id)

  <<***extension***>> (p. *116*) *The RTPS Instance ID of the* ***dds::domain::DomainParticipant*** (p. *1326*).
- uint32_t **rtps_instance_id** () const

  Getter (see the setter with the same name)
- **WireProtocol** & **rtps_well_known_ports** (const **rti::core::RtpsWellKnownPorts** &the_rtps_well_known_↩ ports)

  <<***extension***>> (p. *116*) *Configures the RTPS well-known port mappings.*
- **rti::core::RtpsWellKnownPorts** **rtps_well_known_ports** () const

  Getter (see the setter with the same name)
- **WireProtocol** & **rtps_reserved_port_mask** (const **RtpsReservedPortKindMask** &the_rtps_reserved_port↩ _mask)

  <<***extension***>> (p. *116*) *Specifies which well-known ports to reserve when enabling the participant.*
- **RtpsReservedPortKindMask** **rtps_reserved_port_mask** () const

  Getter (see the setter with the same name)
- **WireProtocol** & **rtps_auto_id_kind** (**rti::core::policy::WireProtocolAutoKind** the_kind)

  <<***extension***>> (p. *116*) *Kind of auto mechanism used to calculate the GUID prefix.*
- **rti::core::policy::WireProtocolAutoKind** **rtps_auto_id_kind** () const

  Getter (see the setter with the same name)
- **WireProtocol** & **compute_crc** (bool crc_needed)

  <<***extension***>> (p. *116*) *Adds RTPS CRC submessage to every message when this field is set to true.*
- bool **compute_crc** () const

  Getter (see the setter with the same name)
- **WireProtocol** & **check_crc** (bool crc_checked)

  <<***extension***>> (p. *116*) *Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to true.*
- bool **check_crc** () const

  Getter (see the setter with the same name)

Static Public Attributes

- static OMG_DDS_API_CLASS_VARIABLE const uint32_t **RTPS_AUTO_ID**

  Indicates that RTI Connext should choose an appropriate host, app, instance or object ID automatically.

8.432.1 Detailed Description

<<***extension***>> *(p. 116)* Configures the write protocol of a **dds::domain::DomainParticipant** (p. 1326)

Entity:

dds::domain::DomainParticipant (p. 1326)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **NO** (p. 514)

8.432.2 Usage

This QoS policy configures some participant-wide properties of the DDS Real-Time Publish Subscribe (RTPS) on-the-wire protocol. (**rti::core::policy::DataWriterProtocol** (p. 1210) and **rti::core::policy::DataReaderProtocol** (p. 1059) configure RTPS and reliability properties on a per **dds::pub::DataWriter** (p. 1138) or **dds::sub::DataReader** (p. 976) basis.)

NOTE: The default QoS policies returned by RTI Connext contain the correctly initialized wire protocol attributes. The defaults are not normally expected to be modified, but are available to the advanced user customizing the implementation behavior.

The default values should not be modified without an understanding of the underlying Real-Time Publish Subscribe (RTPS) wire protocol.

In order for the discovery process to work correctly, each **dds::domain::DomainParticipant** (p. 1326) must have a unique identifier. This QoS policy specifies how that identifier should be generated.

RTPS defines a 96-bit prefix to this identifier; each **dds::domain::DomainParticipant** (p. 1326) must have a unique value for this prefix relative to all other participants in its domain.

If an application dies unexpectedly and is restarted, the IDs used by the new instance of DomainParticipants should be different than the ones used by the previous instances. A change in these values allows other DomainParticipants to know that they are communicating with a new instance of an application, and not the previous instance.

For legacy reasons, RTI Connext divides the 96-bit prefix into three integers:

- The first integer is called host ID. The original purpose of this integer was to contain the identity of the machine on which the DomainParticipant is executing.
- The second integer is called an application ID. The original purpose of this integer was to contain a value that identifies the process or task in which the DomainParticipant is contained.
- The third integer is called instance ID. The original purpose was to contain a value that uniquely identifies a DomainParticipant within a task or process.

The **rti::core::policy::WireProtocol::rtps_auto_id_kind** (p. 2639) field can be used to configure the algorithm that RTI Connext uses to populate the 96-bit prefix. Then you can optionally overwrite specific parts of the 96-bit prefix by explicitly configuring the **rti::core::policy::WireProtocol::rtps_host_id** (p. 2636) (first integer), **rti::core::policy↩ ::WireProtocol::rtps_app_id** (p. 2636) (second integer), and **rti::core::policy::WireProtocol::rtps_instance_id** (p. 2637) (third integer).

The **rti::core::policy::WireProtocol::rtps_auto_id_kind** (p. 2639) field supports three different prefix generation algorithms:

1. In the default and most common scenario, **rti::core::policy::WireProtocol::rtps_auto_id_kind** (p. 2639) is set to **rti::core::policy::WireProtocolAutoKind_def::RTPS_AUTO_ID_FROM_UUID** (p. 2642). As the name suggests, this mechanism uses a unique, randomly generated UUID to fill the **rtps_host_id**, **rtps_app_id**, or **rtps_instance_id** fields. The first two bytes of the **rtps_host_id** are replaced with the RTI vendor ID (0x0101).

2. (Legacy) When rtps_auto_id_kind is set to **rti::core::policy::WireProtocolAutoKind_def::RTPS_AUTO_ID_↩ FROM_IP** (p. 2642), the 96-bit prefix is generated as follows:

   - **rtps_host_id:** The 32-bit value of the IPv4 of the first up and running interface of the host machine is assigned. If the host does not have an IPv4 address, the host-id will be automatically set to 0x7F000001.
   - **rtps_app_id:** The process (or task) ID is assigned.
   - **rtps_instance_id:** A counter is assigned that is incremented per new participant within a process.

   DDS_RTPS_AUTO_ID_FROM_IP is not a good algorithm to guarantee prefix uniqueness, because the process ID can be recycled by the OSs.

3. (Legacy) When rtps_auto_id_kind is set to **rti::core::policy::WireProtocolAutoKind_def::RTPS_AUTO_ID_↩ FROM_MAC** (p. 2642), the 96-bit prefix is generated as follows:

   - **rtps_host_id:** The first 32 bits of the MAC address of the first up and running interface of the host machine are assigned.
   - **rtps_app_id:** The last 32 bits of the MAC address of the first up and running interface of the host machine are assigned.
   - **rtps_instance_id:** This field is split into two different parts. The process (or task) ID is assigned to the first 24 bits. A counter is assigned to the last 8 bits. This counter is incremented per new participant.

   DDS_RTPS_AUTO_ID_FROM_IP is not a good algorithm to guarantee prefix uniqueness, because the process ID can be recycled by the OSs.

Some examples are provided to clarify the behavior of this QoSPolicy in case you want to change the default behavior with **rti::core::policy::WireProtocolAutoKind_def::RTPS_AUTO_ID_FROM_MAC** (p. 2642).

First, get the participant QoS from the DomainParticipantFactory:

Second, change the **rti::core::policy::WireProtocol** (p. 2630) using one of the options shown below.

Third, create the **dds::domain::DomainParticipant** (p. 1326) as usual, using the modified QoS structure instead of the default one.

Option 1: Use **rti::core::policy::WireProtocolAutoKind_def::RTPS_AUTO_ID_FROM_MAC** (p. 2642) to explicitly set just the application/task identifier portion of the **rtps_instance_id** field.

```
using namespace rti::core::policy;
```

```
participant_qos « WireProtocol()

    .rtps_auto_id_kind(WireProtocolAutoKind::RTPS_AUTO_ID_FROM_MAC)

    .rtps_instance_id(

        /* App ID */ (12 « 8) | /* Instance ID */ WireProtocol::RTPS_AUTO_ID);
```

Option 2: Handle only the per participant counter and let RTI Connext handle the application/task identifier:

```
using namespace rti::core::policy;

participant_qos « WireProtocol()

    .rtps_auto_id_kind(WireProtocolAutoKind::RTPS_AUTO_ID_FROM_MAC)

    .rtps_instance_id(

        /* App ID */ WireProtocol::RTPS_AUTO_ID | /* Instance ID */ 12);
```

Option 3: Handle the entire **rtps_instance_id** field yourself:

```
using namespace rti::core::policy;

participant_qos « WireProtocol()

    .rtps_auto_id_kind(WireProtocolAutoKind::RTPS_AUTO_ID_FROM_MAC)

    .rtps_instance_id(

        /* App ID */ (12 « 8) | /* Instance ID */ 9);
```

NOTE: If you are using **rti::core::policy::WireProtocolAutoKind_def::RTPS_AUTO_ID_FROM_MAC** (p. 2642) as **rtps_auto_id_kind** and you decide to manually handle the **rtps_instance_id** field, you must ensure that both parts are non-zero (otherwise RTI Connext will take responsibility for them). RTI recommends that you always specify the two parts separately in order to avoid errors.

Option 4: Let RTI Connext handle the entire **rtps_instance_id** field:

```
using namespace rti::core::policy;

participant_qos « WireProtocol().rtps_auto_id_kind(

    WireProtocolAutoKind::RTPS_AUTO_ID_FROM_MAC);
```

NOTE: If you are using **rti::core::policy::WireProtocolAutoKind_def::RTPS_AUTO_ID_FROM_MAC** (p. 2642) as **rtps_auto_id_kind** and you decide to manually handle the **rtps_instance_id** field, you must ensure that both parts are non-zero (otherwise RTI Connext will take responsibility for them). RTI recommends that you always specify the two parts separately in order to clearly show the difference.

8.432.3 Constructor & Destructor Documentation

8.432.3.1 WireProtocol()

```
rti::core::policy::WireProtocol::WireProtocol ( )  [inline]
```

Creates the default policy.

8.432.4 Member Function Documentation

8.432.4.1 participant_id() [1/2]

```
WireProtocol & rti::core::policy::WireProtocol::participant_id (
            int32_t the_participant_id )
```

<<***basic***>> *(*p. *104)* <<***extension***>> *(*p. *116)* A value used to distinguish among different participants belonging to the same domain on the same host.

Determines the unicast port on which meta-traffic is received. Also defines the *default* unicast port for receiving user-traffic for DataReaders and DataWriters (can be overridden by the **rti::core::policy::TransportUnicast** (p. 2556) or **rti::core::policy::TransportUnicast** (p. 2556)).

For more information on port mapping, please refer to **rti::core::RtpsWellKnownPorts** (p. 2258).

Each **dds::domain::DomainParticipant** (p. 1326) in the same domain and running on the same host, must have a unique `participant_id`. The participants may be in the same address space or in distinct address spaces.

A negative number (-1) means that RTI Connext will *automatically* resolve the participant ID as follows.

- RTI Connext will pick the *smallest* participant ID based on the unicast ports available on the transports enabled for discovery.
- RTI Connext will attempt to resolve an automatic port index either when a DomainPartcipant is enabled, or when a DataReader or DataWriter is created. Therefore, all the transports enabled for discovery must have been registered by this time. Otherwise, the discovery transports registered after resolving the automatic port index may produce port conflicts when the DomainPartcipant is enabled.

[default] -1 [automatic], i.e. RTI Connext will automatically pick the `participant_id`, as described above.

[range] [>= 0], or -1, and does not violate guidelines stated in **rti::core::RtpsWellKnownPorts** (p. 2258).

See also

dds::core::Entity::enable() (p. 1524)

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.432.4.2 participant_id() [2/2]

```
int32_t rti::core::policy::WireProtocol::participant_id ( ) const
```

Getter (see the setter with the same name)

8.432.4.3 rtps_host_id() [1/2]

```
WireProtocol & rti::core::policy::WireProtocol::rtps_host_id (
            uint32_t the_rtps_host_id )
```

<<***extension***>> *(*p. *116)* The RTPS Host ID of the domain participant.

A specific host ID that is unique in the domain.

[default] rti::core::policy::WireProtocol::RTPS_AUTO_ID (p. 2641). The default value is interpreted as follows:
If **rti::core::policy::WireProtocol::rtps_auto_id_kind** (p. 2639) is equal to **rti::core::policy::WireProtocolAuto↩ Kind_def::RTPS_AUTO_ID_FROM_IP** (p. 2642), the value will be interpreted as the IPv4 address of the *first* up and running interface of the host machine.
If **rti::core::policy::WireProtocol::rtps_auto_id_kind** (p. 2639) is equal to **rti::core::policy::WireProtocolAuto↩ Kind_def::RTPS_AUTO_ID_FROM_MAC** (p. 2642), the value will be interpreted as the first 32 bits of the MAC address assigned to the *first* up and running interface of the host machine.
If **rti::core::policy::WireProtocol::rtps_auto_id_kind** (p. 2639) is equal to **rti::core::policy::WireProtocolAuto↩ Kind_def::RTPS_AUTO_ID_FROM_UUID** (p. 2642), the value will be the first 32 bits of the GUID prefix assigned by the UUID algorithm.

[range] [0,0xffffffff]

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.432.4.4 rtps_host_id() [2/2]

```
uint32_t rti::core::policy::WireProtocol::rtps_host_id ( ) const
```

Getter (see the setter with the same name)

8.432.4.5 rtps_app_id() [1/2]

```
WireProtocol & rti::core::policy::WireProtocol::rtps_app_id (
            uint32_t the_rtps_app_id )
```

<<***extension***>> *(*p. *116)* The RTPS App ID of the domain participant.

A participant specific ID that, together with the rtps_instance_id, is unique within the scope of the rtps_host_id.

If a participant dies and is restarted, it is recommended that it be given an app ID that is distinct from the previous one so that other participants in the domain can distinguish between them.

[default] rti::core::policy::WireProtocol::RTPS_AUTO_ID (p. 2641). The default value is interpreted as follows:
If **rti::core::policy::WireProtocol::rtps_auto_id_kind** (p. 2639) is equal to ***rti::core::policy::WireProtocolAuto↩ Kind_def::RTPS_AUTO_ID_FROM_IP*** *(*p. *2642)* the value will be the process (or task) ID.
If **rti::core::policy::WireProtocol::rtps_auto_id_kind** (p. 2639) is equal to ***rti::core::policy::WireProtocolAuto↩ Kind_def::RTPS_AUTO_ID_FROM_MAC*** *(*p. *2642)* the value will be the last 32 bits of the MAC address assigned to the *first* up and running interface of the host machine.
If **rti::core::policy::WireProtocol::rtps_auto_id_kind** (p. 2639) is equal to ***rti::core::policy::WireProtocolAuto↩ Kind_def::RTPS_AUTO_ID_FROM_UUID*** *(*p. *2642)*, the value will be the middle 32 bits of the GUID prefix assigned by the UUID algorithm.

[range] [0,0xffffffff]

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.432.4.6 rtps_app_id() [2/2]

```
uint32_t rti::core::policy::WireProtocol::rtps_app_id ( ) const
```

Getter (see the setter with the same name)

8.432.4.7 rtps_instance_id() [1/2]

```
WireProtocol & rti::core::policy::WireProtocol::rtps_instance_id (
            uint32_t the_rtps_instance_id )
```

<<***extension***>> *(*p. *116)* The RTPS Instance ID of the **dds::domain::DomainParticipant** (p. 1326).

This is an instance-specific ID of a participant that, together with the rtps_app_id, is unique within the scope of the rtps_host_id.

If a participant dies and is restarted, it is recommended that it be given an instance ID that is distinct from the previous one so that other participants in the domain can distinguish between them.

[default] rti::core::policy::WireProtocol::RTPS_AUTO_ID (p. 2641). The default value is interpreted as follows:
If **rti::core::policy::WireProtocol::rtps_auto_id_kind** (p. 2639) is equal to ***rti::core::policy::WireProtocolAuto↩ Kind_def::RTPS_AUTO_ID_FROM_IP*** *(*p. *2642)*, a counter is assigned that is incremented per new participant. For VxWorks-653, the first 8 bits are assigned to the partition id for the application.
If **rti::core::policy::WireProtocol::rtps_auto_id_kind** (p. 2639) is equal to ***rti::core::policy::WireProtocolAuto↩ Kind_def::RTPS_AUTO_ID_FROM_MAC*** *(*p. *2642)*, the first 24 bits are assigned to the application/task identifier and the last 8 bits are assigned to a counter that is incremented per new paricipant.
If **rti::core::policy::WireProtocol::rtps_auto_id_kind** (p. 2639) is equal to ***rti::core::policy::WireProtocolAuto↩ Kind_def::RTPS_AUTO_ID_FROM_UUID*** *(*p. *2642)*, the value will be the last 32 bits of the GUID prefix assigned by the UUID algorithm.

[range] [0,0xffffffff] **NOTE:** If you use ***rti::core::policy::WireProtocolAutoKind_def::RTPS_AUTO_ID_FROM_MAC*** *(*p. *2642)* as **rtps_auto_id_kind** and you decide to manually handle the **rtps_instance_id** field, you must ensure that both the two parts are non-zero, otherwise the middleware will take responsibility for them. We recommend that you always specify the two parts separately in order to avoid errors. (`examples`)

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.432.4.8 rtps_instance_id() [2/2]

```
uint32_t rti::core::policy::WireProtocol::rtps_instance_id ( ) const
```

Getter (see the setter with the same name)

8.432.4.9 rtps_well_known_ports() [1/2]

```
WireProtocol & rti::core::policy::WireProtocol::rtps_well_known_ports (
            const  rti::core::RtpsWellKnownPorts & the_rtps_well_known_ports )
```

<<***extension***>> *(*p. *116)* Configures the RTPS well-known port mappings.

Determines the well-known multicast and unicast port mappings for discovery (meta) traffic and user traffic.

[default] rti::core::RtpsWellKnownPorts::Interoperable() (p. 557)

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.432.4.10 rtps_well_known_ports() [2/2]

```
rti::core::RtpsWellKnownPorts rti::core::policy::WireProtocol::rtps_well_known_ports ( ) const
```

Getter (see the setter with the same name)

8.432.4.11 rtps_reserved_port_mask() [1/2]

```
WireProtocol & rti::core::policy::WireProtocol::rtps_reserved_port_mask (
            const  RtpsReservedPortKindMask & the_rtps_reserved_port_mask )
```

<<***extension***>> *(p. 116)* Specifies which well-known ports to reserve when enabling the participant.

Specifies which of the well-known multicast and unicast ports will be reserved when the DomainParticipant is enabled. Failure to allocate a port that is computed based on the **rti::core::RtpsWellKnownPorts** (p. 2258) will be detected at this time, and the enable operation will fail.

[default] RTPS_RESERVED_PORT_MASK_DEFAULT

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.432.4.12 rtps_reserved_port_mask() [2/2]

```
RtpsReservedPortKindMask rti::core::policy::WireProtocol::rtps_reserved_port_mask ( ) const
```

Getter (see the setter with the same name)

8.432.4.13 rtps_auto_id_kind() [1/2]

```
WireProtocol & rti::core::policy::WireProtocol::rtps_auto_id_kind (
            rti::core::policy::WireProtocolAutoKind the_kind )
```

<<***extension***>> *(p. 116)* Kind of auto mechanism used to calculate the GUID prefix.

[default] rti::core::policy::WireProtocolAutoKind_def::RTPS_AUTO_ID_FROM_UUID (p. 2642)

Categories

Immutable (p. 243), **Discovery** (p. 177)

8.432.4.14 rtps_auto_id_kind() [2/2]

```
rti::core::policy::WireProtocolAutoKind rti::core::policy::WireProtocol::rtps_auto_id_kind ( )
const
```

Getter (see the setter with the same name)

8.432.4.15 compute_crc() [1/2]

```
WireProtocol & rti::core::policy::WireProtocol::compute_crc (
            bool crc_needed )
```

<<***extension***>> *(*p. *116)* Adds RTPS CRC submessage to every message when this field is set to true.

The computed CRC covers the entire RTPS message excluding the RTPS header.

[default] false

Categories

Immutable (p. 243), **Security** (p. 235)

8.432.4.16 compute_crc() [2/2]

```
bool rti::core::policy::WireProtocol::compute_crc ( ) const
```

Getter (see the setter with the same name)

8.432.4.17 check_crc() [1/2]

```
WireProtocol & rti::core::policy::WireProtocol::check_crc (
            bool crc_checked )
```

<<***extension***>> *(*p. *116)* Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to true.

rti::core::policy::WireProtocol::compute_crc (p. 2640) must be enabled at the publishing application for validating the message at the subscribing application.

[default] false

Categories

Immutable (p. 243), **Security** (p. 235)

8.432.4.18 check_crc() [2/2]

```
bool rti::core::policy::WireProtocol::check_crc ( ) const
```

Getter (see the setter with the same name)

8.432.5 Member Data Documentation

8.432.5.1 RTPS_AUTO_ID

```
OMG_DDS_API_CLASS_VARIABLE const uint32_t rti::core::policy::WireProtocol::RTPS_AUTO_ID  [static]
```

Indicates that RTI Connext should choose an appropriate host, app, instance or object ID automatically.

If this special value is assigned to **rti::core::policy::WireProtocol::rtps_host_id** (p. 2636), **rti::core::policy::WireProtocol::rtps_app_id** (p. 2636), **rti::core::policy::WireProtocol::rtps_instance_id** (p. 2637), **rti::core::policy::DataWriterProtocol::rtps_object_id** (p. 1212) or **rti::core::policy::DataReaderProtocol::rtps_object_id** (p. 1061) RTI Connext will assign the ID automatically.

The actual ID value is chosen when the QoS is set: the QoS returned from **dds::domain::DomainParticipant::qos() const** (p. 1336), **dds::pub::DataWriter::qos() const** (p. 1166)or **dds::sub::DataReader::qos() const** (p. 1007)will never have this value.

QoS:

rti::core::policy::WireProtocol::rtps_host_id (p. 2636) **rti::core::policy::WireProtocol::rtps_app_id** (p. 2636) **rti::core::policy::WireProtocol::rtps_instance_id** (p. 2637)

8.433 rti::core::policy::WireProtocolAutoKind_def Struct Reference

*<<**extension**>> (p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) WireProtocolAutoKind

```
#include <PolicyKind.hpp>
```

Public Types

- enum **type** {
  RTPS_AUTO_ID_FROM_IP ,
  RTPS_AUTO_ID_FROM_MAC ,
  RTPS_AUTO_ID_FROM_UUID }

  The underlying `enum` type.

8.433.1 Detailed Description

<<***extension***>> *(p. 354)* The definition of the **dds::core::safe_enum** (p. 2265) WireProtocolAutoKind

8.433.2 Member Enumeration Documentation

8.433.2.1 type

`enum` **rti::core::policy::WireProtocolAutoKind_def::type**

The underlying `enum` type.

Enumerator

RTPS_AUTO_ID_FROM_IP	Builds the GUID prefix using the IPv4 address and process ID. Uses the IPv4 address of the first up-and-running interface of the host machine and the process ID to build the GUID prefix.
RTPS_AUTO_ID_FROM_MAC	Builds the GUID prefix using the MAC address and process ID. Uses the first 32 bits of the MAC address assigned to the first up-and-running interface and the process ID to build the GUID prefix. Note to Android Users: DDS_RTPS_AUTO_ID_FROM_MAC is not supported in recent versions of Android (6.0 and later).
RTPS_AUTO_ID_FROM_UUID	Builds the GUID prefix by selecting the UUID-based algorithm (default). This method of generating the GUID prefix does not require having a network interface and is friendly to IP mobility scenarios in which an RTI Connext application may start in a node that does not have a physical network interface enabled.

8.434 rti::pub::WriteParams Class Reference

<<***extension***>> *(p. 354)* <<***value-type***>> *(p. 350)* Advanced parameters for writing with a DataWriter

`#include "rti/pub/WriteParams.hpp"`

Public Member Functions

- **WriteParams** ()

  *Creates a **WriteParams** (p. 2642) object with default values.*
- void **reset** ()

  Resets all the fields to their default values.
- bool **replace_automatic_values** () const

Indicates if the replacement of automatic values has been activated or not.

- **WriteParams** & **replace_automatic_values** (bool value)

  Allows retrieving the actual value of those fields that were automatic.
- **rti::core::SampleIdentity identity** () const

  Gets the sample identity.
- **WriteParams** & **identity** (const **rti::core::SampleIdentity** &value)

  Sets a explicit sample identity for writing.
- **rti::core::SampleIdentity related_sample_identity** () const

  Gets the related sample identity.
- **WriteParams** & **related_sample_identity** (const **rti::core::SampleIdentity** &value)

  Sets a related sample identity for a sample.
- **dds::core::Time source_timestamp** () const

  Gets the timestamp.
- **WriteParams** & **source_timestamp** (const **dds::core::Time** &value)

  Sets the source timestamp for writing.
- **rti::core::Cookie cookie** () const

  Gets the cookie.
- **WriteParams** & **cookie** (const **rti::core::Cookie** &value)

  Sets a cookie for writing.
- **dds::core::InstanceHandle handle** () const

  Gets the instance handle.
- **WriteParams** & **handle** (const **dds::core::InstanceHandle** &value)

  Sets the instance handle for writing.
- int32_t **priority** () const

  Gets the priority.
- **WriteParams** & **priority** (int32_t value)

  Sets the priority for writing.
- **rti::core::SampleFlag flag** () const

  Gets the sample flag.
- **WriteParams** & **flag** (const **rti::core::SampleFlag** &value)

  Sets the sample flags for writing.
- **rti::core::Guid source_guid** () const

  Gets the source GUID.
- **WriteParams** & **source_guid** (const **rti::core::Guid** &value)

  Sets the source GUID for writing.
- **rti::core::Guid related_source_guid** () const

  Gets the related source guid.
- **WriteParams** & **related_source_guid** (const **rti::core::Guid** &value)

  Sets the related source guid for writing.
- **rti::core::Guid related_reader_guid** () const

  Gets the related reader GUID.
- **WriteParams** & **related_reader_guid** (const **rti::core::Guid** &value)

  Sets the related reader guid for writing.

8.434.1 Detailed Description

*<<**extension**>> (p. 354) <<**value-type**>> (p. 350)* Advanced parameters for writing with a DataWriter

See also

dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&) (p. 1179)

8.434.2 Constructor & Destructor Documentation

8.434.2.1 WriteParams()

```
rti::pub::WriteParams::WriteParams ( )  [inline]
```

Creates a **WriteParams** (p. 2642) object with default values.

The default instance is such that it makes **dds::pub::DataWriter(const T&, WriteParams&)** (p. 1138) behave identically to **dds::pub::DataWriter(const T&)** (p. 1138).

8.434.3 Member Function Documentation

8.434.3.1 reset()

```
void rti::pub::WriteParams::reset ( )
```

Resets all the fields to their default values.

See also

replace_automatic_values() (p. 2644)

8.434.3.2 replace_automatic_values() [1/2]

```
bool rti::pub::WriteParams::replace_automatic_values ( ) const
```

Indicates if the replacement of automatic values has been activated or not.

8.434.3.3 replace_automatic_values() [2/2]

```
WriteParams & rti::pub::WriteParams::replace_automatic_values (
            bool value )
```

Allows retrieving the actual value of those fields that were automatic.

When this field is set, the fields that were configured with an automatic value (for example, **rti::core::SampleIdentity::automatic()** (p. 2284)) receive their actual value after **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179) is called.

To reset those fields to their automatic value after calling **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179), use **rti::pub::WriteParams::reset** (p. 2644)

Parameters

value	Whether to replace automatic values or not

[default] false

8.434.3.4 identity() [1/2]

```
rti::core::SampleIdentity rti::pub::WriteParams::identity ( ) const
```

Gets the sample identity.

8.434.3.5 identity() [2/2]

```
WriteParams & rti::pub::WriteParams::identity (
            const  rti::core::SampleIdentity & value )
```

Sets a explicit sample identity for writing.

Identifies the sample being written. The identity consists of a pair (Virtual Writer GUID, Virtual Sequence Number).

Use the default value to let RTI Connext determine the sample identity as follows:

- The Virtual Writer GUID is the virtual GUID associated with the writer writing the sample. This virtual GUID is configured using **rti::core::policy::DataWriterProtocol::virtual_guid** (p. 1212).
- The sequence number is increased by one with respect to the previous value.

The virtual sequence numbers for a virtual writer must be strictly monotonically increasing. If the user tries to write a sample with a sequence number smaller or equal to the last sequence number, the write operation will fail.

A DataReader can access the identity of a received sample by using the fields **dds::sub::SampleInfo::original_publication_virtual_guid** (p. 2291) and **dds::sub::SampleInfo::original_publication_virtual_sequence_number** (p. 2291) in the **dds::sub::SampleInfo** (p. 2285).

[default] rti::core::SampleIdentity::automatic() (p. 2284).

8.434.3.6 related_sample_identity() [1/2]

```
rti::core::SampleIdentity rti::pub::WriteParams::related_sample_identity ( ) const
```

Gets the related sample identity.

8.434.3.7 related_sample_identity() [2/2]

```
WriteParams & rti::pub::WriteParams::related_sample_identity (
            const  rti::core::SampleIdentity & value )
```

Sets a related sample identity for a sample.

Identifies another sample that is logically related to the one that is written.

When this field is set, the related sample identity is propagated and subscribing applications can retrieve it from the **dds::sub::SampleInfo** (p. 2285) (see **dds::sub::SampleInfo::related_original_publication_virtual_sample_identity** (p. 2293)).

The default value is **rti::core::SampleIdentity::unknown()** (p. 2284), and is not propagated.

A DataReader can access the related identity of a received sample by using the fields **dds::sub::SampleInfo::related_original_publication_virtual_guid** (p. 2292) and **dds::sub::SampleInfo::related_original_publication_virtual_sequence_number** (p. 2292) in the **dds::sub::SampleInfo** (p. 2285).

[default] rti::core::SampleIdentity::unknown() (p. 2284)

8.434.3.8 source_timestamp() [1/2]

```
dds::core::Time rti::pub::WriteParams::source_timestamp ( ) const
```

Gets the timestamp.

8.434.3.9 source_timestamp() [2/2]

```
WriteParams & rti::pub::WriteParams::source_timestamp (
            const  dds::core::Time & value )
```

Sets the source timestamp for writing.

Specifies the source timestamp that will be available to the **dds::sub::DataReader** (p. 976) objects by means of the `source_timestamp` attribute within the **dds::sub::SampleInfo** (p. 2285).

[default] dds::core::Time::invalid() (p. 2459).

8.434.3.10 cookie() [1/2]

```
rti::core::Cookie rti::pub::WriteParams::cookie ( ) const
```

Gets the cookie.

8.434.3.11 cookie() [2/2]

```
WriteParams & rti::pub::WriteParams::cookie (
            const  rti::core::Cookie & value )
```

Sets a cookie for writing.

Used in the callback **dds::pub::DataWriterListener::on_sample_removed** (p. 1209) to associate a removed sample with a written sample.

[default] Empty sequence (zero-length).

8.434.3.12 handle() [1/2]

```
dds::core::InstanceHandle rti::pub::WriteParams::handle ( ) const
```

Gets the instance handle.

Either the handle returned by a previous call to **dds::pub::DataWriter::register_instance** (p. 1158), or else the special value **dds::core::InstanceHandle::nil()** (p. 1614).

[default] dds::core::InstanceHandle::nil() (p. 1614)

8.434.3.13 handle() [2/2]

```
WriteParams & rti::pub::WriteParams::handle (
            const  dds::core::InstanceHandle & value )
```

Sets the instance handle for writing.

ă Instance handle

Either the handle returned by a previous call to **dds::pub::DataWriter::register_instance** (p. 1158), or else the special value **dds::core::InstanceHandle::nil()** (p. 1614).

[default] dds::core::InstanceHandle::nil() (p. 1614)

8.434.3.14 priority() [1/2]

```
int32_t rti::pub::WriteParams::priority ( ) const
```

Gets the priority.

8.434.3.15 priority() [2/2]

```
WriteParams & rti::pub::WriteParams::priority (
            int32_t value )
```

Sets the priority for writing.

A positive integer value designating the relative priority of the sample, used to determine the transmission order of pending writes.

Use of publication priorities requires an asynchronous publisher (**rti::core::policy::PublishModeKind_def::ASYNCHRONOUS** (p. 2019)) with **rti::pub::FlowControllerProperty::scheduling_policy** (p. 1579) set to **rti::core::policy::FlowControllerSchedulingPolicy_def::HIGHEST_PRIORITY_FIRST** (p. 1583).

Larger numbers have higher priority.

For multi-channel DataWriters, the publication priority of a sample may be used as a filter criteria for determining channel membership.

If the publication priority of the parent DataWriter, or for multi-channel DataWriters, if the publication priority of the parent channel, is set to PUBLICATION_PRIORITY_AUTOMATIC, then the DataWriter or channel will be assigned the priority of the largest publication priority of all samples in the DataWriter or channel.

If the publication priority of the parent DataWriter, and for multi-channel DataWriters, if the publication priority of the parent channel, are set to PUBLICATION_PRIORITY_UNDEFINED, then the DataWriter or channel will be assigned the lowest priority, regardless of the value of the publication priorities of samples written to the DataWriter or channel.

The publication priority of each sample can be set in the **rti::pub::WriteParams** (p. 2642) of **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179).

For dispose and unregister samples, use the **rti::pub::WriteParams** (p. 2642) of **dds::pub::DataWriter::dispose_instance(rti::pub::WriteParams&)** (p. 1173) and **dds::pub::DataWriter::unregister_instance(rti::pub::WriteParams&)** (p. 1173).

[default] 0 (lowest priority)

See also

rti::core::ChannelSettings::priority (p. 924)

8.434.3.16 flag() [1/2]

```
rti::core::SampleFlag rti::pub::WriteParams::flag ( ) const
```

Gets the sample flag.

8.434.3.17 flag() [2/2]

```
WriteParams & rti::pub::WriteParams::flag (
            const  rti::core::SampleFlag & value )
```

Sets the sample flags for writing.

The flags are represented as a 32-bit integer, of which only the 16 least-significant bits are used.

RTI reserves least-significant bits [0-7] for middleware-specific usage.

The application can use least-significant bits [8-15].

The first bit (**rti::core::SampleFlag::redelivered** (p. 2280)) is reserved for marking samples as redelivered when using RTI Queuing Service.

The second bit (**rti::core::SampleFlag::intermediate_reply_sequence** (p. 2280)) is used to indicate that a response sample is not the last response sample for a given request. This bit is usually set by a Replier sending multiple responses for a request.

An application can inspect the flags associated with a received sample by checking the field **dds::sub::SampleInfo↩ ::flag** (p. 2293).

[default] 0 (no flags are set)

8.434.3.18 source_guid() [1/2]

```
rti::core::Guid rti::pub::WriteParams::source_guid ( ) const
```

Gets the source GUID.

8.434.3.19 source_guid() [2/2]

```
WriteParams & rti::pub::WriteParams::source_guid (
            const  rti::core::Guid & value )
```

Sets the source GUID for writing.

When this field is set, the source_guid is propagated and subscribing applications can retrieve it from the **dds::sub::↩ SampleInfo** (p. 2285) (see **dds::sub::SampleInfo::source_guid** (p. 2293)).

The default value is **rti::core::Guid::automatic()** (p. 1597), and is not propagated.

This field is used by RTI Connext in the following cases:

- In RTI Queuing Service, this field and **rti::pub::WriteParams::related_source_guid** (p. 2651) play a crucial role in ensuring that replies are correctly routed back to the requester in request/reply scenarios.
- In RTI Routing Service, this field is used by the TopicQuery proxy feature to identify the original writer(s) that published the response samples for the TopicQuery. When dealing with snapshot TopicQueries, there will be one final sample (sample without the flag **rti::core::SampleFlag::intermediate_topic_query_sample** (p. 2281)) per original writer that published one or more response samples.

You can use this field for your own purposes unless you are using Queuing Service or Routing Service as described above.

[default] rti::core::Guid::automatic() (p. 1597) (the source_guid is automatically set to the **dds::pub::DataWriter** (p. 1138) virtual GUID).

See also

rti::pub::WriteParams::related_source_guid (p. 2651)

8.434.3.20 related_source_guid() [1/2]

```
rti::core::Guid rti::pub::WriteParams::related_source_guid ( ) const
```

Gets the related source guid.

8.434.3.21 related_source_guid() [2/2]

```
WriteParams & rti::pub::WriteParams::related_source_guid (
            const  rti::core::Guid & value )
```

Sets the related source guid for writing.

When this field is set, the related_source_guid is propagated and subscribing applications can retrieve it from the **dds::sub::SampleInfo** (p. 2285) (see **dds::sub::SampleInfo::related_source_guid** (p. 2294)).

This field and **rti::pub::WriteParams::source_guid** (p. 2650) are used by RTI Queuing Service, to ensure that replies are correctly routed back to the requester in request/reply scenarios.

The default value is **rti::core::Guid::unknown()** (p. 1597), and is not propagated.

[default] rti::core::Guid::unknown() (p. 1597)

See also

rti::pub::WriteParams::source_guid (p. 2650)

8.434.3.22 related_reader_guid() [1/2]

```
rti::core::Guid rti::pub::WriteParams::related_reader_guid ( ) const
```

Gets the related reader GUID.

8.434.3.23 related_reader_guid() [2/2]

```
WriteParams & rti::pub::WriteParams::related_reader_guid (
            const  rti::core::Guid & value )
```

Sets the related reader guid for writing.

When this field is set, the related_reader_guid is propagated and subscribing applications can retrieve it from the **dds::sub::SampleInfo** (p. 2285) (see **dds::sub::SampleInfo::related_subscription_guid** (p. 2294)).

The default value is **rti::core::Guid::unknown()** (p. 1597), and is not propagated.

The main use case for this field is point-to-point sample distribution using CFT. DataReaders install a CFT on the related_reader_guid using a unique GUID. For example, the filter for DataReader 'n' can be:

```
@related_reader_guid.value = &hex(00000000000000000000000000000001)
```

Then, a DataWriter that wants to send the sample to DataReader 'n' will use the **dds::pub::DataWriter::write(const T&,rti::pub::WriteParams&)** (p. 1179) method and set related_reader_guid to the value used by DataReader 'n' in its filter expression.

This field is currently used by RTI Queuing Service to distribute a sample to only one of the Consumer's DataReaders attached to a SharedReaderQueue.

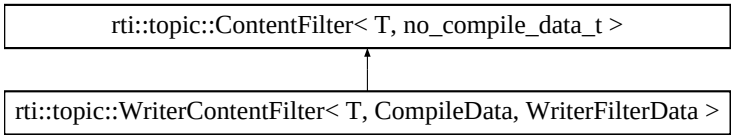
[default] rti::core::Guid::unknown() (p. 1597)

8.435 rti::topic::WriterContentFilter< T, CompileData, WriterFilterData > Class Template Reference

<<***extension***>> *(p. 354)* A class to inherit from when implementing a writer-side custom content filter

```
#include <rti/topic/ContentFilter.hpp>
```

Inheritance diagram for rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >:



Public Member Functions

- virtual void **writer_compile** (WriterFilterData &writer_filter_data, **ExpressionProperty** &prop, const std::string &expression, const **dds::core::StringSeq** &parameters, const **dds::core::optional**< **dds::core::xtypes::DynamicType** > &type_code, const std::string &type_class_name, const **rti::core::Cookie** &cookie)=0

  A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching ***dds::sub::DataReader*** *(*p. *976).*
- virtual rti::core::CookieSeq & **writer_evaluate** (WriterFilterData &writer_filter_data, const T &sample, const **FilterSampleInfo** &meta_data)=0

  A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching ***dds::sub::DataReader*** *(*p. *976).*
- virtual void **writer_finalize** (WriterFilterData &writer_filter_data, const **rti::core::Cookie** &cookie)=0

  A writer-side filtering API to clean up a previously compiled instance of the content filter.
- virtual WriterFilterData & **writer_attach** ()=0

  A writer-side filtering API to create some state that can facilitate filtering on the writer side.
- virtual void **writer_detach** (WriterFilterData &writer_filter_data)=0

  A writer-side filtering API to clean up a previously created state using writer_attach.
- virtual void **writer_return_loan** (WriterFilterData &writer_filter_data, rti::core::CookieSeq &cookies)=0

  A writer-side filtering API to return the loan on the list of DataReaders returned by writer_evaluate.

Additional Inherited Members

8.435.1 Detailed Description

template<typename T, typename CompileData = no_compile_data_t, typename WriterFilterData = no_compile_data_t>
class rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >

<<***extension***>> *(*p. *354)* A class to inherit from when implementing a writer-side custom content filter

This interface can be implemented by an application-provided class and then registered with the DomainParticipant such that samples can be filtered for **dds::topic::ContentFilteredTopic** (p. 954) with the filter name that the filter is registered with. This class inherits from **rti::topic::ContentFilter** (p. 950), therefore you must implement the compile, evaluate, and finalize funtions from that class in addition to the functions in this class.

Note: the API for using a custom content filter is subject to change in a future release.

For an example of how to create a custom content filter see **Creating Custom Content Filters** (p. 333)

Template Parameters

T	The type of the samples that this **ContentFilter** (p. 950) will filter
CompileData	the type of the data that the compile function will return. If your compile function will not return data, you can leave this template parameter to the default **no_compile_data_t** (p. 1835) type and return rti::topic::no_compile_data in your compile function
WriterFilterData	the type of the data that the writer_attach function will return. If your writer_attach function will not return data, you can leave this template parameter to the default **no_compile_data_t** (p. 1835) type and return rti::topic::no_compile_data in your compile function

See also

rti::topic::CustomFilter (p. 967)

rti::topic::ContentFilter (p. 950)

dds::domain::DomainParticipant::register_contentfilter() (p. 1349)

8.435.2 Member Function Documentation

8.435.2.1 writer_compile()

```
template<typename T , typename CompileData = no_compile_data_t, typename WriterFilterData = no_↩
compile_data_t>
virtual void  rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >::writer_compile
(
            WriterFilterData & writer_filter_data,
             ExpressionProperty & prop,
            const std::string & expression,
            const  dds::core::StringSeq & parameters,
            const  dds::core::optional<  dds::core::xtypes::DynamicType > & type_code,
            const std::string & type_class_name,
            const  rti::core::Cookie & cookie )  [pure virtual]
```

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching **dds::sub::DataReader** (p. 976).

This method is called when the **dds::pub::DataWriter** (p. 1138) discovers a **dds::sub::DataReader** (p. 976) with a **dds::topic::ContentFilteredTopic** (p. 954) or when a **dds::pub::DataWriter** (p. 1138) is notified of a change in a DataReader's filter parameter for the locally registered content filter instance.

It is possible for multiple threads to be calling into this function at the same time.

Parameters

writer_filter_data	The state created using writer_attach
prop	An **ExpressionProperty** (p. 1547) that allows you to indicate to RTI Connext if a filter expression can be optimized.
expression	An std::string with the filter expression. If you want to manipulate this string, you **must** first make a copy of it.
parameters	A string sequence with the expression parameters the ContentFilteredTopic was created with. The string sequence is **equal** (but **not** identical) to the string sequence passed to **dds::topic::ContentFilteredTopic()** (p. 954). Note that the sequence passed to the compile function is **owned** by RTI Connext and **must not** be referenced outside the compile function.
type_code	The DynamicType for the related Topic of the ContentFilteredTopic. A type code is a description of a type in terms of which **types** it contains (int32, string, etc.) and the corresponding member field names in the data type structure. The type code can be used to write custom content filters that can be used with any type.
type_class_name	Fully qualified class name of the related Topic.
cookie	**rti::core::Cookie** (p. 965) to uniquely identify the DataReader for which writer_compile was called

8.435.2.2 writer_evaluate()

```
template<typename T , typename CompileData = no_compile_data_t, typename WriterFilterData = no_↩
compile_data_t>
virtual rti::core::CookieSeq &  rti::topic::WriterContentFilter< T, CompileData, WriterFilterData
>::writer_evaluate (
            WriterFilterData & writer_filter_data,
            const T & sample,
            const  FilterSampleInfo & meta_data )  [pure virtual]
```

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching **dds::sub::DataReader** (p. 976).

This method is called every time a **dds::pub::DataWriter** (p. 1138) writes a new sample. Its purpose is to evaluate the sample for all the readers for which the **dds::pub::DataWriter** (p. 1138) is performing writer-side filtering and return the list of **rti::core::Cookie** (p. 965) structures associated with the DataReaders whose filters pass the sample.

It is possible for multiple threads to be calling into this function at the same time

Parameters

writer_filter_data	The state created using the writer_compile method
sample	A deserialized sample to be filtered.
meta_data	Meta data associated with the sample.

Returns

A rti::core::CookieSeq which identifies the set of DataReaders whose filters pass the sample.

8.435.2.3 writer_finalize()

```
template<typename T , typename CompileData = no_compile_data_t, typename WriterFilterData = no_↩
compile_data_t>
virtual void  rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >::writer_↩
finalize (
            WriterFilterData & writer_filter_data,
            const  rti::core::Cookie & cookie )  [pure virtual]
```

A writer-side filtering API to clean up a previously compiled instance of the content filter.

This method is called to notify the filter implementation that the **dds::pub::DataWriter** (p. 1138) is no longer matching with a **dds::sub::DataReader** (p. 976) for which it was previously performing writer-side filtering. This will allow the filter to purge any state it was maintaining for the **dds::sub::DataReader** (p. 976).

It is possible for multiple threads to be calling into this function at the same time.

Parameters

writer_filter_data	The state created using writer_attach
cookie	**rti::core::Cookie** (p. 965) to uniquely identify the DataReader for which this method was invoked

8.435.2.4 writer_attach()

```
template<typename T , typename CompileData = no_compile_data_t, typename WriterFilterData = no_↩
compile_data_t>
virtual WriterFilterData &  rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >↩
::writer_attach ( )  [pure virtual]
```

A writer-side filtering API to create some state that can facilitate filtering on the writer side.

This method is called to create some state required to perform filtering on the writer side using writer-side filtering APIs. This method will be called for every **dds::pub::DataWriter** (p. 1138); it will be called only the first time the **dds::pub::↩ DataWriter** (p. 1138) matches a **dds::sub::DataReader** (p. 976) using the specified filter. This function will not be called for any subsequent DataReaders that match the DataWriter and are using the same filter.

Returns

An object of user-specified type to some state created on the DataWriter that will help perform writer-side filtering efficiently. Can be no_compile_data

8.435.2.5 writer_detach()

```
template<typename T , typename CompileData = no_compile_data_t, typename WriterFilterData = no_↩
compile_data_t>
virtual void  rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >::writer_detach
(
            WriterFilterData & writer_filter_data )  [pure virtual]
```

A writer-side filtering API to clean up a previously created state using writer_attach.

This method is called to delete any state created using the **WriterContentFilter::writer_attach** (p. 2656) function. This method will be called when the **dds::pub::DataWriter** (p. 1138) is deleted.

Parameters

writer_filter_data	The state created using writer_attach

8.435.2.6 writer_return_loan()

```
template<typename T , typename CompileData = no_compile_data_t, typename WriterFilterData = no_↩
compile_data_t>
virtual void  rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >::writer_↩
return_loan (
            WriterFilterData & writer_filter_data,
            rti::core::CookieSeq & cookies )  [pure virtual]
```

A writer-side filtering API to return the loan on the list of DataReaders returned by writer_evaluate.

This method is called to return the loan on CookieSeq returned by **WriterContentFilter::writer_return_loan** (p. 2656). It is possible for multiple threads to be calling into this function at the same time.

Parameters

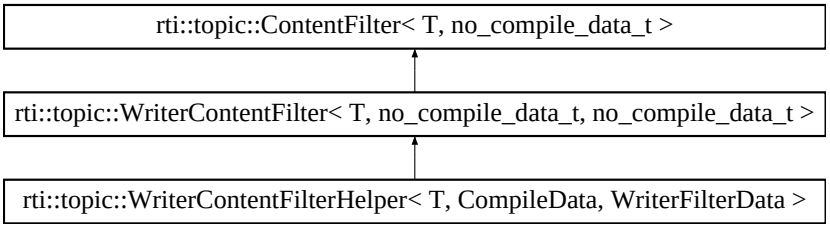
writer_filter_data	The state created using writer_attach
cookies	rti::core::CookieSeq for which writer_return_loan was invoked

8.436 rti::topic::WriterContentFilterHelper< T, CompileData, WriterFilterData > Class Template Reference

<<***extension***>> *(p. 354)* A class to inherit from when implementing a writer-side custom content filter.

```
#include <rti/topic/ContentFilter.hpp>
```

Inheritance diagram for rti::topic::WriterContentFilterHelper< T, CompileData, WriterFilterData >:



Public Member Functions

- virtual void **writer_evaluate_helper** (WriterFilterData &writer_filter_data, const T &sample, const **Filter↩ SampleInfo** &meta_data)=0

  *A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching **dds::sub::DataReader** (p. 976).*

Protected Member Functions

- void **add_cookie** (**rti::core::Cookie** &cookie)

  *A helper function which will add a **rti::core::Cookie** (p. 965) to the CookieSeq that is then returned by the writer_evaluate function.*

Additional Inherited Members

8.436.1 Detailed Description

template<typename T, typename CompileData = no_compile_data_t, typename WriterFilterData = no_compile_data_t>
class rti::topic::WriterContentFilterHelper< T, CompileData, WriterFilterData >

<<***extension***>> *(p. 354)* A class to inherit from when implementing a writer-side custom content filter.

This class manages for you the rti::core::CookieSeq which maintains a list of Cookies associated with the DataReaders whose filters pass the sample.

This class has a private CookieSeq member that you can add cookies too using the helper method, **add_cookie()** (p. 2659), and the sequence is automatically cleared for you in the return_loan method, meaning that you do not need to implement that method yourself.

For an example of how to create a custom content filter see **Creating Custom Content Filters** (p. 333)

Template Parameters

T	The type of the samples that this **ContentFilter** (p. 950) will filter
CompileData	the type of the data that the compile function will return. If your compile function will not return data, you can leave this template parameter to the default **no_compile_data_t** (p. 1835) type and return rti::topic::no_compile_data in your compile function
WriterFilterData	the type of the data that the writer_attach function will return. If your writer_attach function will not return data, you can leave this template parameter to the default **no_compile_data_t** (p. 1835) type and return rti::topic::no_compile_data in your compile function

See also

rti::topic::CustomFilter (p. 967)

rti::topic::WriterContentFilter (p. 2652)

rti::topic::ContentFilter (p. 950)

dds::domain::DomainParticipant::register_contentfilter() (p. 1349)

8.436.2 Member Function Documentation

8.436.2.1 writer_evaluate_helper()

```
template<typename T , typename CompileData = no_compile_data_t, typename WriterFilterData = no_↩
compile_data_t>
virtual void  rti::topic::WriterContentFilterHelper< T, CompileData, WriterFilterData >::writer↩
_evaluate_helper (
            WriterFilterData & writer_filter_data,
            const T & sample,
            const  FilterSampleInfo & meta_data )  [pure virtual]
```

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching **dds::sub::DataReader** (p. 976).

This method is called every time a **dds::pub::DataWriter** (p. 1138) writes a new sample. Its purpose is to evaluate the sample for all the readers for which the **dds::pub::DataWriter** (p. 1138) is performing writer-side filtering and return the list of **rti::core::Cookie** (p. 965) structures associated with the DataReaders whose filters pass the sample.

It is possible for multiple threads to be calling into this function at the same time

When using the **WriterContentFilterHelper** (p. 2657) helper class, you implement this method instead of writer_↩ evaluate. The writer_evaluate function calls this method and then returns the CookieSeq for you.

Parameters

writer_filter_data	The state created using the writer_compile method
sample	A deserialized sample to be filtered.
meta_data	Meta data associated with the sample.

8.436.2.2 add_cookie()

```
template<typename T , typename CompileData = no_compile_data_t, typename WriterFilterData = no_↩
compile_data_t>
void  rti::topic::WriterContentFilterHelper< T, CompileData, WriterFilterData >::add_cookie (
            rti::core::Cookie & cookie )  [inline], [protected]
```

A helper function which will add a **rti::core::Cookie** (p. 965) to the CookieSeq that is then returned by the writer_↩ evaluate function.

Call this method from writer_evaluate_helper whenever you need to add a Cookie for a DataReader whose filter has passed a particular sample.

Parameters

cookie	The cookie to add to the CookieSeq

8.437 dds::core::policy::WriterDataLifecycle Class Reference

Controls how a **dds::pub::DataWriter** (p. 1138) handles the lifecycle of the instances (keys) that it writes.

`#include <dds/core/policy/CorePolicy.hpp>`

Public Member Functions

- **WriterDataLifecycle** ()

  Creates the default policy.
- **WriterDataLifecycle** (bool autodispose)

  Creates an instance with a value for auto-dispose unregistered instances and default values for the rest of parameters.
- **WriterDataLifecycle** & **autodispose_unregistered_instances** (bool b)

  Indicates whether the DataWriter should automatically dispose an instance when it unregisters it.
- bool **autodispose_unregistered_instances** () const

  Getter (see setter with the same name)
- **dds::core::policy::WriterDataLifecycle** & **autopurge_unregistered_instances_delay** (const **dds::core::Duration** &duration)

  *<<**extension**>> (p. 116) Maximum duration for which the DataWriter will maintain information regarding an instance once it has unregistered the instance.*
- **dds::core::Duration autopurge_unregistered_instances_delay** () const

  *<<**extension**>> (p. 116) Getter (see setter with the same name)*
- **dds::core::policy::WriterDataLifecycle** & **autopurge_disposed_instances_delay** (const **dds::core::Duration** &duration)

  *<<**extension**>> (p. 116) Maximum duration for which the DataWriter will maintain information regarding an instance once it has disposed the instance.*
- **dds::core::Duration autopurge_disposed_instances_delay** () const

  *<<**extension**>> (p. 116) Getter (see setter with the same name)*

Static Public Member Functions

- static **WriterDataLifecycle AutoDisposeUnregisteredInstances** ()

  Creates `WriterDataLifecycle(true)`
- static **WriterDataLifecycle ManuallyDisposeUnregisteredInstances** ()

  Creates `WriterDataLifecycle(false)`

8.437.1 Detailed Description

Controls how a **dds::pub::DataWriter** (p. 1138) handles the lifecycle of the instances (keys) that it writes.

Entity:

dds::pub::DataWriter (p. 1138)

Properties:

RxO (p. 514) = N/A
Changeable (p. 514) = **YES** (p. 514)

8.437.2 Usage

This policy determines how the **dds::pub::DataWriter** (p. 1138) acts with regards to the lifecycle of the data instances it manages (data instances that have been either explicitly registered with the **dds::pub::DataWriter** (p. 1138) or implicitly registered by directly writing the data).

You may use **dds::pub::DataWriter::unregister_instance** (p. 1160) to indicate that the **dds::pub::DataWriter** (p. 1138) no longer wants to send data for a **dds::topic::Topic** (p. 2474).

The behavior controlled by this QoS policy applies on a per instance (key) basis for keyed Topics, so that when a **dds↩ ::pub::DataWriter** (p. 1138) unregisters an instance, RTI Connext can automatically also dispose that instance. This is the default behavior.

In many cases where the ownership of a Topic is shared (see **dds::core::policy::Ownership** (p. 1899)), DataWriters may want to relinquish their ownership of a particular instance of the Topic to allow other DataWriters to send updates for the value of that instance regardless of **Ownership** (p. 1899) Strength. In that case, you may only want a DataWriter to unregister an instance without disposing the instance. *Disposing* an instance is a statement that an instance no longer exists. User applications may be coded to trigger on the disposal of instances, thus the ability to unregister without disposing may be useful to properly maintain the semantic of disposal.

8.437.3 Constructor & Destructor Documentation

8.437.3.1 WriterDataLifecycle() [1/2]

```
dds::core::policy::WriterDataLifecycle::WriterDataLifecycle ( )  [inline]
```

Creates the default policy.

8.437.3.2 WriterDataLifecycle() [2/2]

```
dds::core::policy::WriterDataLifecycle::WriterDataLifecycle (
            bool autodispose )  [inline], [explicit]
```

Creates an instance with a value for auto-dispose unregistered instances and default values for the rest of parameters.

8.437.4 Member Function Documentation

8.437.4.1 autodispose_unregistered_instances() [1/2]

```
WriterDataLifecycle & dds::core::policy::WriterDataLifecycle::autodispose_unregistered_instances
(
            bool b )  [inline]
```

Indicates whether the DataWriter should automatically dispose an instance when it unregisters it.

- true
  The **dds::pub::DataWriter** (p. 1138) will dispose of the instance each time it is unregistered. The behavior is identical to explicitly calling one of the `dispose` operations on the instance prior to calling the `unregister` operation.
- false (default)
  The **dds::pub::DataWriter** (p. 1138) will not dispose of the instance. The application can still call one of the dispose operations prior to unregistering the instance and dispose of the instance that way.

[default] false

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193), **Sample Memory Management** (p. 230)

8.437.4.2 autodispose_unregistered_instances() [2/2]

```
bool dds::core::policy::WriterDataLifecycle::autodispose_unregistered_instances ( ) const  [inline]
```

Getter (see setter with the same name)

8.437.4.3 AutoDisposeUnregisteredInstances()

```
static  WriterDataLifecycle dds::core::policy::WriterDataLifecycle::AutoDisposeUnregistered↩
Instances ( )  [inline], [static]
```

Creates `WriterDataLifecycle(true)`

8.437.4.4 ManuallyDisposeUnregisteredInstances()

```
static  WriterDataLifecycle dds::core::policy::WriterDataLifecycle::ManuallyDisposeUnregistered↩
Instances ( )  [inline], [static]
```

Creates `WriterDataLifecycle(false)`

8.437.4.5 autopurge_unregistered_instances_delay() [1/2]

```
dds::core::policy::WriterDataLifecycle & autopurge_unregistered_instances_delay (
            const  dds::core::Duration & duration )
```

<<***extension***>> *(*p. *116)* Maximum duration for which the DataWriter will maintain information regarding an instance once it has unregistered the instance.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Determines how long the **dds::pub::DataWriter** (p. 1138) will maintain information regarding an instance that has been unregistered. By default, the **dds::pub::DataWriter** (p. 1138) resources associated with an instance (e.g., the space needed to remember the Instance Key or KeyHash) are released lazily. This means the resources are only reclaimed when the space is needed for another instance because **dds::core::policy::ResourceLimits::max_instances** (p. 2206) is exceeded. This behavior can be changed by setting autopurge_unregistered_instances_delay to a value other than **dds::core::Duration::infinite()** (p. 1451).

After this time elapses, the **dds::pub::DataWriter** (p. 1138) will purge all internal information regarding the instance, including historical samples, even if **dds::core::policy::ResourceLimits::max_instances** (p. 2206) has not been reached.

The purging of unregistered instances can be done based on the source timestamp of the unregister sample or the time where the unregister sample was added to the DataWriter queue by setting the following property to 1 or 0 respectively (default: 0): **dds.data_writer.history.source_timestamp_based_autopurge_instances_delay**.

For durable writer history, autopurge_unregistered_instances_delay supports only the **dds::core::Duration::infinite()** (p. 1451) value.

With **dds::core::policy::DurabilityKind_def::VOLATILE** (p. 1442) durability, unregistered instances are removed as soon as all samples for the instance are fully acknowledged, regardless of how this field is set.

[default] dds::core::Duration::infinite() (p. 1451) (disabled) for all **dds::pub::DataWriter** (p. 1138) except for the built-in discovery DataWriters
dds::core::Duration::zero() (p. 1450) for built-in discovery DataWriters (see **rti::core::policy::DiscoveryConfig**↩ **::publication_writer_data_lifecycle** (p. 1291), **rti::core::policy::DiscoveryConfig::subscription_writer_data**↩ **_lifecycle** (p. 1293) and **rti::core::policy::DiscoveryConfig::participant_configuration_writer_data_lifecycle** (p. 1316)).

[range] [0, 1 year] or **dds::core::Duration::infinite()** (p. 1451)

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193), **Sample Memory Management** (p. 230)

8.437.4.6 autopurge_unregistered_instances_delay() [2/2]

```
dds::core::Duration autopurge_unregistered_instances_delay ( ) const
```

<<***extension***>> *(p. 116)* Getter (see setter with the same name)

8.437.4.7 autopurge_disposed_instances_delay() [1/2]

```
dds::core::policy::WriterDataLifecycle & autopurge_disposed_instances_delay (
            const  dds::core::Duration & duration )
```

<<***extension***>> *(p. 116)* Maximum duration for which the DataWriter will maintain information regarding an instance once it has disposed the instance.

Note

This function is an extension, it must be called via the **extensions() member function** (p. 354)

Determines how long the **dds::pub::DataWriter** (p. 1138) will maintain information regarding an instance that has been disposed of. By default, disposing of an instance does not make it eligible to be purged. By setting autopurge_↩ disposed_instances_delay to a value other than **dds::core::Duration::infinite()** (p. 1451), the DataWriter will delete the resources associated with an instance (including historical samples) once the time has elapsed and all matching DataReaders have acknowledged all the samples for this instance including the dispose sample.

The purging of disposed instances can be done based on the source timestamp of the dispose sample or the time when the dispose sample was added to the DataWriter queue by setting the following property to 1 or 0 respectively (default: 0): **dds.data_writer.history.source_timestamp_based_autopurge_instances_delay**.

This QoS value is supported with durable DataWriter queues only for **dds::core::Duration::zero()** (p. 1450) and **dds**↩ **::core::Duration::infinite()** (p. 1451) values (finite values are not supported).

[default] dds::core::Duration::infinite() (p. 1451) (disabled)

[range] [0, 1 year] or **dds::core::Duration::infinite()** (p. 1451)

Categories

Immutable (p. 243), **Instance Lifecycle** (p. 193), **Sample Memory Management** (p. 230)

8.437.4.8 autopurge_disposed_instances_delay() [2/2]

```
dds::core::Duration autopurge_disposed_instances_delay ( ) const
```

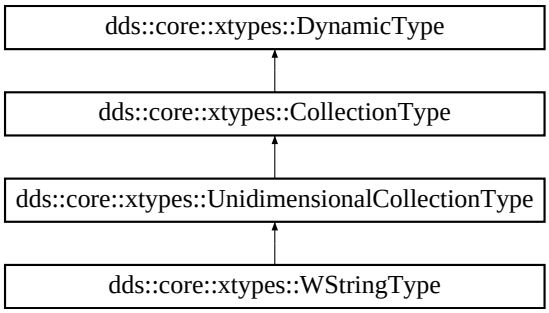
<<***extension***>> *(p. 116)* Getter (see setter with the same name)

8.438 dds::core::xtypes::WStringType Class Reference

<<***value-type***>> *(p. 350)* Represents an IDL `wstring` type.

`#include <dds/core/xtypes/CollectionTypes.hpp>`

Inheritance diagram for dds::core::xtypes::WStringType:



Public Member Functions

- **WStringType** (uint32_t **bounds**)
  Creates a bounded wide string.

Additional Inherited Members

8.438.1 Detailed Description

<<***value-type***>> *(p. 350)* Represents an IDL `wstring` type.

8.438.2 Constructor & Destructor Documentation

8.438.2.1 WStringType()

```
dds::core::xtypes::WStringType::WStringType (
            uint32_t bounds ) [explicit]
```

Creates a bounded wide string.

Parameters

bounds	The maximum length

8.439 rti::config::compliance::XTypesMask Class Reference

This mask configures different aspects of the Extended CDR encoding (XCDR and XCDR2).

`#include <rti/config/Compliance.hpp>`

Inherits std::bitset< 32 >.

Public Types

- typedef std::bitset< 32 > **MaskType**

  A typedef of std::bitset<32> for convenience.

Public Member Functions

- **XTypesMask** ()

  Default XTypes compliance mask.
- **XTypesMask** (uint64_t mask)

  *Construct a **XTypesMask** (p. 2666) from an integer.*
- **XTypesMask** (const **MaskType** &mask)

  *Construct a **XTypesMask** (p. 2666) from a MaskType object.*

Static Public Member Functions

- static const **XTypesMask dheader_in_non_primitive_collections** ()

  When this bit is set, the serialization of sequences and arrays with non-primitive members includes a DHEADER.
- static const **XTypesMask enum_as_primitive_in_collections** ()

  When this bit is set, enums are considered primitive types in collection types.
- static const **XTypesMask parameter_length_with_padding** ()

  When this bit is set, the length of a member header in a mutable type or an optional member will include padding bytes to a 4-byte boundary.
- static const **XTypesMask encapsulation_options_with_padding** ()

  When this bit is set, RTI Connext will set the padding bits in the options field of the encapsulation header of a serialized payload.
- static const **XTypesMask accept_unknown_enum_value** ()

  When this bit is set, samples containing an unknown enumerator can be successfully deserialized to the default enumeration value.
- static const **XTypesMask accept_unknown_discriminator** ()

  When this bit is set, samples containing an unknown discriminator can be successfully deserialized, and the sample won't be dropped.
- static const **XTypesMask select_default_discriminator** ()

  When this bit is set, samples containing a discriminator value that does not select any member in the union will be assigned the default discriminator value; if the bit is not set, the sample will preserve the discriminator value.
- static const **XTypesMask sentinel_in_empty_union** ()

  When this bit is set, samples of a mutable union that contain an unknown discriminator will be serialized with a 4-byte sentinel value.

- static const **XTypesMask initialize_discriminator_to_default** ()

  When this bit is set, the union discriminator will be initialized to the default value of its type.
- static const **XTypesMask java_keyhash_750** ()

  When this bit is set, the KeyHash calculation for a Java instance will match the one produced by the Java API in Connext 7.5.0 and earlier versions.
- static const **XTypesMask python_int8_730** ()

  Set this bit to interoperate with Connext 7.3.0 Python applications that published or subscribed to types with int8 members.
- static const **XTypesMask must_understand_in_key_and_discriminator** ()

  When this bit is set, the parameter ID for members that are part of the key, or for the discriminator of a union, will have the parameter ID's FLAG_MUST_UNDERSTAND bit set when serialized using XCDR2.
- static const **XTypesMask vendor** ()

  XTypes vendor compliance mask. This value is fully aligned with the XTypes specification.
- static const **XTypesMask default_mask** ()

  Default XTypes compliance mask.

8.439.1 Detailed Description

This mask configures different aspects of the Extended CDR encoding (XCDR and XCDR2).

XTypes compliance mask.

The **rti::config::compliance::XTypesMask** (p. 2666) configures different aspects of the Extended CDR encoding (XCDR and XCDR2).

Applications that don't use the exact same XTypes compliance mask may not be interoperable.

By default, data serialization is not fully compliant with Extended CDR encoding. While this doesn't affect correctness, it may prevent interoperability with other vendors.

You can change the default compliance mask for an application using any of the following mechanisms:

- Setting the environment variable **NDDS_XTYPES_COMPLIANCE_MASK**
- Calling the function **rti::config::compliance::set_xtypes_mask()** (p. 452)

The environment variable will be automatically loaded when a DomainParticipantFactory is created. If you need to load it before that, use the function **rti::config::compliance::load_compliance_masks()** (p. 453).

The format of the environment variable is an unsigned integer, which can be in HEX notation. For example: 0x00000001.

The default value is **rti::config::compliance::XTypesMask::default_mask()** (p. 2674).

To be fully compliant with the XTypes specification, set the mask to **rti::config::compliance::XTypesMask::vendor()** (p. 2674).

[default] rti::config::compliance::XTypesMask::default_mask() (p. 2674)

When changing the mask for an application is not desired, the property dds.xtypes.compliance_mask will change the mask at the DomainParticipant or DataWriter level. When set in the DomainParticipant, it configures the mask for all DataWriters in the participant. See `Extensible Types Compliance Mask` in the RTI Connext Core Libraries Extensible Types Guide.

8.439.2 Member Typedef Documentation

8.439.2.1 MaskType

```
typedef std::bitset<32>  rti::config::compliance::XTypesMask::MaskType
```

A typedef of std::bitset<32> for convenience.

8.439.3 Constructor & Destructor Documentation

8.439.3.1 XTypesMask() [1/3]

```
rti::config::compliance::XTypesMask::XTypesMask ( )  [inline]
```

Default XTypes compliance mask.

The mask sets the following bits:

- **rti::config::compliance::XTypesMask::parameter_length_with_padding()** (p. 2669)
- **rti::config::compliance::XTypesMask::encapsulation_options_with_padding()** (p. 2670)
- **rti::config::compliance::XTypesMask::sentinel_in_empty_union()** (p. 2671)
- **rti::config::compliance::XTypesMask::initialize_discriminator_to_default()** (p. 2672)

The HEX value for the mask is: 0x0000018C.

8.439.3.2 XTypesMask() [2/3]

```
rti::config::compliance::XTypesMask::XTypesMask (
            uint64_t mask )  [inline], [explicit]
```

Construct a **XTypesMask** (p. 2666) from an integer.

Parameters

mask	Value whose bits are copied to the bitset positions

8.439.3.3 XTypesMask() [3/3]

```
rti::config::compliance::XTypesMask::XTypesMask (
            const  MaskType & mask )  [inline]
```

Construct a **XTypesMask** (p. 2666) from a MaskType object.

Parameters

mask	A std::bitset<32> to construct this **XTypesMask** (p. 2666) from

8.439.4 Member Function Documentation

8.439.4.1 dheader_in_non_primitive_collections()

```
static const  XTypesMask rti::config::compliance::XTypesMask::dheader_in_non_primitive_collections
( )  [inline], [static]
```

When this bit is set, the serialization of sequences and arrays with non-primitive members includes a DHEADER.

Value: 0x00000001

To be compatible with the XTypes specification, this bit must be set.

This bit only applies to XCDR2 encoding.

Limitation: For FlatData, the compliant encoding is applied to arrays and sequences of mutable types, but not final types . The serialization of final types containing arrays of non-primitive types in the FlatData language binding is not compliant with Extended CDR encoding version 2 in any case.

This bit is unset by default. See **rti::config::compliance::XTypesMask::default_mask()** (p. 2674)

8.439.4.2 enum_as_primitive_in_collections()

```
static const  XTypesMask rti::config::compliance::XTypesMask::enum_as_primitive_in_collections ( )
[inline], [static]
```

When this bit is set, enums are considered primitive types in collection types.

Value: 0x00000002

A DHEADER will not be added to collections of enums if **rti::config::compliance::XTypesMask::dheader_in_non_↩ primitive_collections()** (p. 2669) is set.

To be compatible with the XTypes specification, this bit must be unset.

This bit only applies to XCDR2 encoding.

This bit is unset by default. See **rti::config::compliance::XTypesMask::default_mask()** (p. 2674)

8.439.4.3 parameter_length_with_padding()

```
static const XTypesMask rti::config::compliance::XTypesMask::parameter_length_with_padding ( )
[inline], [static]
```

When this bit is set, the length of a member header in a mutable type or an optional member will include padding bytes to a 4-byte boundary.

Value: 0x00000004

To be compatible with the XTypes specification, this bit must be unset.

This bit only applies to XCDR encoding.

This bit is set by default. See **rti::config::compliance::XTypesMask::default_mask()** (p. 2674)

8.439.4.4 encapsulation_options_with_padding()

```
static const XTypesMask rti::config::compliance::XTypesMask::encapsulation_options_with_padding (
) [inline], [static]
```

When this bit is set, RTI Connext will set the padding bits in the options field of the encapsulation header of a serialized payload.

Value: 0x00000008

When this bit is set RTI Connext will set the least significant two bits in the second byte of the options field to a value that encodes the number of padding bytes needed after the end of the serialized payload in order to reach the next 4-byte aligned offset. Specifically, the least significant two bits shall be set to binary 00 if no padding bits would be needed and binary 01, 10, or 11 if, respectively, there would be one, two, or three bytes of padding needed. These bits in the options field will be interpreted by a DataReader to determine where the serialized data exactly ended.

To be compatible with the XTypes specification, this bit must be set.

To be compatible with the following Micro versions this bit must be unset because those versions of Micro did not support padding bits and will discard any data with encapsulations options not set to 0.

- RTI Connext Micro 3.x.y.z
- RTI Connext Micro 2.4.15.1
- RTI Connext Micro 2.4.14.0 and 2.4.14.1
- RTI Connext Micro 2.4.13.1 through 2.4.13.5
- RTI Connext Micro 2.4.12.z

This bit applies to both, XCDR and XCDR2 encodings.

This bit is set by default. See **rti::config::compliance::XTypesMask::default_mask()** (p. 2674)

8.439.4.5 accept_unknown_enum_value()

```
static const XTypesMask rti::config::compliance::XTypesMask::accept_unknown_enum_value ( ) [inline],
[static]
```

When this bit is set, samples containing an unknown enumerator can be successfully deserialized to the default enumeration value.

Value: 0x00000010

To be compatible with the XTypes specification, this bit must be unset.

This bit is unset by default. See **rti::config::compliance::XTypesMask::default_mask()** (p. 2674)

8.439.4.6 accept_unknown_discriminator()

```
static const XTypesMask rti::config::compliance::XTypesMask::accept_unknown_discriminator ( )
[inline], [static]
```

When this bit is set, samples containing an unknown discriminator can be successfully deserialized, and the sample won't be dropped.

Value: 0x00000020

When this bit is set, samples containing an unknown discriminator won't be dropped, and the discriminator value assigned to the sample will depend on whether or not the bit **rti::config::compliance::XTypesMask::select_default_discriminator()** (p. 2671) is set.

To be compatible with the XTypes specification, this bit must be set.

This bit is unset by default. See **rti::config::compliance::XTypesMask::default_mask()** (p. 2674)

8.439.4.7 select_default_discriminator()

```
static const XTypesMask rti::config::compliance::XTypesMask::select_default_discriminator ( )
[inline], [static]
```

When this bit is set, samples containing a discriminator value that does not select any member in the union will be assigned the default discriminator value; if the bit is not set, the sample will preserve the discriminator value.

Value: 0x00000040

When this bit is set, samples containing a discriminator value that does not select any member in the union will be assigned the default discriminator value. If the default value selects a valid union member, that member will be initialized to the default value of its type. If the bit is not set, the sample will preserve the discriminator value. The bit **rti::config::compliance::XTypesMask::initialize_discriminator_to_default()** (p. 2672) determines what the default discriminator value is.

This bit is only relevant when the bit **rti::config::compliance::XTypesMask::accept_unknown_discriminator()** (p. 2671) is set, and it is used to determine the value of the discriminator when the sample contains an unknown discriminator.

To be compatible with the XTypes specification, this bit must be unset.

This bit is unset by default. See **rti::config::compliance::XTypesMask::default_mask()** (p. 2674)

8.439.4.8 sentinel_in_empty_union()

```
static const XTypesMask rti::config::compliance::XTypesMask::sentinel_in_empty_union ( ) [inline],
[static]
```

When this bit is set, samples of a mutable union that contain an unknown discriminator will be serialized with a 4-byte sentinel value.

Value: 0x00000080

When this bit is set, samples of a mutable union that contain an unknown discriminator will be serialized with a 4-byte sentinel value: 0x027f0000. This is the same sentinel value as the one used for samples of a mutable union that contain a known discriminator.

This bit only applies to XCDR encoding, Dynamic Data, and language APIs other than Java. When using Java and not using Dynamic Data, the sentinel is always added.

To be compatible with the XTypes specification and with Connext version 5.3.x, this bit must be set.

This bit is set by default. See **rti::config::compliance::XTypesMask::default_mask()** (p. 2674)

8.439.4.9 initialize_discriminator_to_default()

```
static const XTypesMask rti::config::compliance::XTypesMask::initialize_discriminator_to_default
( ) [inline], [static]
```

When this bit is set, the union discriminator will be initialized to the default value of its type.

Value: 0x00000100

When this bit is set, the union discriminator will be initialized to the default value of its type. This means 0 for numeric discriminators, false for boolean discriminators, and the first literal of the enumeration for enumerated discriminators. As a result, a union may be initialized with a discriminator value that does not correspond to any of its members.

The default value of the type can be modified by creating an alias type and using the **[default]** annotation to specify the desired default value. You can then use that alias type as the discriminator type, and the default discriminator value will be the one specified in the alias type.

When this bit is unset, the discriminator will be set to select to a value that selects the default union member, if one is defined, or otherwise to the lowest value associated with any member.

If you generated code (using Code Generator) with this bit configured one way using the -xTypesComplianceMask flag (for example, -xTypesComplianceMask 0x00000100) and then configure it the opposite way in the API (for example, unset at 0x00000000), you must regenerate code using Code Generator's -xTypesComplianceMask flag configured the same way as it is in the API (for example, -xTypesComplianceMask 0x00000000). Note that this bit is the only bit that Code Generator uses; therefore, you have to set it the same both in the API and in Code Generator. See information about Code Generator's -xTypesComplianceMask command-line option in `Command-Line Arguments for rtiddsgen`, in the RTI Code Generator User's Manual.

This bit affects **rti::config::compliance::XTypesMask::select_default_discriminator()** (p. 2671). (See that bit's description above.)

To be compatible with the XTypes specification, this bit must be set.

This bit is set by default. See **rti::config::compliance::XTypesMask::default_mask()** (p. 2674)

8.439.4.10 java_keyhash_750()

```
static const XTypesMask rti::config::compliance::XTypesMask::java_keyhash_750 ( ) [inline],
[static]
```

When this bit is set, the KeyHash calculation for a Java instance will match the one produced by the Java API in Connext 7.5.0 and earlier versions.

Value: 0x00000200

When this bit is set, the KeyHash calculation for a Java instance will match the one produced by the Java API in Connext 7.5.0 and earlier versions.

This affects KeyHash calculation only when using XCDR encoding and the Java language API for mutable types and either of the following conditions is met:

- The key is not a primitive type, and the maximum serialized size of a sample for the type exceeds 65KB.
- At least one key member has a member ID greater than 16128.

To be compatible with the XTypes specification, this bit must be unset.

This bit is unset by default. See **rti::config::compliance::XTypesMask::default_mask()** (p. 2674)

8.439.4.11 python_int8_730()

```
static const XTypesMask rti::config::compliance::XTypesMask::python_int8_730 ( ) [inline], [static]
```

Set this bit to interoperate with Connext 7.3.0 Python applications that published or subscribed to types with int8 members.

Value: 0x00000400

When this bit is set, the TypeCode generation for a Python type that contains an int8 member will match the one produced by the Python API in Connext 7.3.0. However, this behavior is not compliant with the XTypes specification, and Python types with int8 members won't interoperate with other APIs.

When this bit is unset, the TypeCode generation for a Python type that contains an int8 member will be done according to the XTypes specification, allowing interoperability with all the APIs except Python in Connext 7.3.0.

To be compatible with the XTypes specification, this bit must be unset.

This bit is unset by default. See **rti::config::compliance::XTypesMask::default_mask()** (p. 2674)

8.439.4.12 must_understand_in_key_and_discriminator()

```
static const XTypesMask rti::config::compliance::XTypesMask::must_understand_in_key_and_discriminator
( ) [inline], [static]
```

When this bit is set, the parameter ID for members that are part of the key, or for the discriminator of a union, will have the parameter ID's FLAG_MUST_UNDERSTAND bit set when serialized using XCDR2.

Value: 0x00000800

To be compatible with the XTypes specification, this bit must be set.

This bit only applies when you are using XCDR2. (For XCDR, the parameter ID's FLAG_MUST_UNDERSTAND bit will never be set for key members and union discriminators, because doing so would change the key hash computation and disrupt communication with older versions of RTI Connext.)

This bit is unset by default. See **rti::config::compliance::XTypesMask::default_mask()** (p. 2674)

8.439.4.13 vendor()

```
static const XTypesMask rti::config::compliance::XTypesMask::vendor ( ) [inline], [static]
```

XTypes vendor compliance mask. This value is fully aligned with the XTypes specification.

The mask sets the following bits:

- **rti::config::compliance::XTypesMask::dheader_in_non_primitive_collections()** (p. 2669)
- **rti::config::compliance::XTypesMask::encapsulation_options_with_padding()** (p. 2670)
- **rti::config::compliance::XTypesMask::accept_unknown_discriminator()** (p. 2671)
- **rti::config::compliance::XTypesMask::sentinel_in_empty_union()** (p. 2671)
- **rti::config::compliance::XTypesMask::initialize_discriminator_to_default()** (p. 2672)

The HEX value for the mask is: 0x000001a9.

Warning

> This mask will always include all known fixes to ensure full alignment with the XTypes specification. As a result, its contents may change between releases or patches. If you want to ensure that your application's behavior remains unaffected by future changes, you should specify the exact bits you are interested in.

8.439.4.14 default_mask()

```
static const XTypesMask rti::config::compliance::XTypesMask::default_mask ( ) [inline], [static]
```

Default XTypes compliance mask.

The mask sets the following bits:

- **rti::config::compliance::XTypesMask::parameter_length_with_padding()** (p. 2669)
- **rti::config::compliance::XTypesMask::encapsulation_options_with_padding()** (p. 2670)
- **rti::config::compliance::XTypesMask::sentinel_in_empty_union()** (p. 2671)
- **rti::config::compliance::XTypesMask::initialize_discriminator_to_default()** (p. 2672)

The HEX value for the mask is: 0x0000018C.

Chapter 9

Example Documentation

9.1 Foo.idl

The definition of the type in IDL that rtiddsgen uses to generate Foo_publisher.cxx, Foo_subscriber.cxx and the type-support code. There is also the definition of MyType and MyOtherType, used in some **Programming How-To's** (p. 362).

See also

Working with IDL types (p. 606)

```
struct Foo {
    long x; //@key
    long y;
};
struct MyType {
    long my_long;
    string<512> my_string;
    Foo my_foo;
    sequence<long, 10> my_sequence;
    Foo my_array[5];
    Foo my_optional; //@Optional
};
enum Color {
    RED,
    GREEN,
    BLUE
};
enum MyUnionDiscriminator {
    USE_LONG,
    USE_STRING,
    USE_FOO
};
union MyUnion switch (MyUnionDiscriminator) {
case USE_LONG:
    long my_long;
case USE_STRING:
    string my_string;
case USE_FOO:
    Foo my_foo;
};
struct MyOtherType {
    long m1;
    double m2;
    string m3;
};
```

9.2 Foo.hpp

The definition of the C++ types create from Foo.idl

See also

Working with IDL types (p. 606)

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from Foo.idl
using RTI Code Generator (rtiddsgen) version 4.7.0.
The rtiddsgen tool is part of the RTI Connext DDS distribution.
For more information, type 'rtiddsgen -help' at a command shell
or consult the Code Generator User's Manual.
*/
#ifndef Foo_982570024_hpp
#define Foo_982570024_hpp
#include <iosfwd>
#if defined(NDDS_USER_DLL_EXPORT) && defined(RTI_WIN32)
#undef RTIUSERDllExport
#define RTIUSERDllExport __declspec(dllexport)
#endif
#if !defined(RTI_WIN32) && defined(NDDS_USER_SYMBOL_EXPORT)
#undef RTIUSERDllExport
#define RTIUSERDllExport __attribute__((visibility("default")))
#endif
#include "dds/core/SafeEnumeration.hpp"
#include "dds/core/String.hpp"
#include "dds/core/array.hpp"
#include "dds/core/vector.hpp"
#include "dds/core/External.hpp"
#include "rti/core/LongDouble.hpp"
#include "rti/core/Pointer.hpp"
#include "rti/core/array.hpp"
#include "rti/topic/TopicTraits.hpp"
#include "omg/types/string_view.hpp"
#include "omg/types/sequence.hpp"
#include "omg/types/optional.hpp"
#ifndef NDDS_STANDALONE_TYPE
#include "cdr/cdr_typeCode.h"
#include "dds/domain/DomainParticipant.hpp"
#include "dds/topic/TopicTraits.hpp"
#include "dds/core/xtypes/DynamicType.hpp"
#include "dds/core/xtypes/StructType.hpp"
#include "dds/core/xtypes/UnionType.hpp"
#include "dds/core/xtypes/EnumType.hpp"
#include "dds/core/xtypes/AliasType.hpp"
#include "rti/util/StreamFlagSaver.hpp"
#include "rti/domain/PluginSupport.hpp"
#endif
#if defined(NDDS_USER_DLL_EXPORT) || defined(NDDS_USER_SYMBOL_EXPORT)
#undef RTIUSERDllExport
#define RTIUSERDllExport
#endif
#if defined(NDDS_USER_DLL_EXPORT) && defined(RTI_WIN32)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport __declspec(dllexport)
#endif
#if !defined(RTI_WIN32) && defined(NDDS_USER_SYMBOL_EXPORT)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport __attribute__((visibility("default")))
#endif
struct NDDSUSERDllExport Foo {
    int32_t x {};
    int32_t y {};
    Foo();
    Foo(int32_t x_,int32_t y_);
};
NDDSUSERDllExport bool operator == (const Foo& a, const Foo& b);
NDDSUSERDllExport bool operator != (const Foo& a, const Foo& b);
NDDSUSERDllExport void swap(Foo& a, Foo& b)  noexcept;
NDDSUSERDllExport std::ostream& operator<<(std::ostream& o, const Foo& sample);
#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
// On Windows, dll-export template instantiations of standard types used by
```

```
// other dll-exported types
template class NDDSUSERDllExport std::allocator< int32_t >;
template class NDDSUSERDllExport std::vector< int32_t >;
#endif
struct NDDSUSERDllExport MyType {
    int32_t my_long {};
    std::string my_string {};
    ::Foo my_foo {};
    ::omg::types::bounded_sequence< int32_t, 10L > my_sequence {};
    ::dds::core::array< ::Foo, 5L> my_array {};
    ::omg::types::optional< ::Foo > my_optional {};
    MyType();
    MyType(int32_t my_long_,const ::omg::types::string_view& my_string_,const ::Foo& my_foo_,const
       ::omg::types::bounded_sequence< int32_t, 10L >& my_sequence_,const ::dds::core::array< ::Foo, 5L>&
       my_array_,const ::omg::types::optional< ::Foo >& my_optional_);
};
NDDSUSERDllExport bool operator == (const MyType& a, const MyType& b);
NDDSUSERDllExport bool operator != (const MyType& a, const MyType& b);
NDDSUSERDllExport void swap(MyType& a, MyType& b)  noexcept;
NDDSUSERDllExport std::ostream& operator<<(std::ostream& o, const MyType& sample);
enum class Color {
    RED,
    GREEN,
    BLUE
};
NDDSUSERDllExport std::ostream& operator << (std::ostream& o,const Color& sample);
enum class MyUnionDiscriminator {
    USE_LONG,
    USE_STRING,
    USE_FOO
};
NDDSUSERDllExport std::ostream& operator << (std::ostream& o,const MyUnionDiscriminator& sample);
class NDDSUSERDllExport MyUnion {
  public:
    MyUnion();
    ::MyUnionDiscriminator& _d()  {
        return m_d_;
    }
    const ::MyUnionDiscriminator& _d() const  {
        return m_d_;
    }
    void _d(::MyUnionDiscriminator value) {
        m_d_ = value;
    }
    int32_t& my_long()  {
        if (_d() != (MyUnionDiscriminator::USE_LONG) ) {
            throw ::dds::core::PreconditionNotMetError(
                "::MyUnion::my_long not selected by the discriminator" );
        }
        return m_u_.my_long;
    }
    const int32_t& my_long() const  {
        if (_d() != (MyUnionDiscriminator::USE_LONG) ) {
            throw ::dds::core::PreconditionNotMetError(
                "::MyUnion::my_long not selected by the discriminator" );
        }
        return m_u_.my_long;
    }
    void my_long(int32_t value) {
        m_u_.my_long = value;
        m_d_= (MyUnionDiscriminator::USE_LONG);
    }
    std::string& my_string()  {
        if (_d() != (MyUnionDiscriminator::USE_STRING) ) {
            throw ::dds::core::PreconditionNotMetError(
                "::MyUnion::my_string not selected by the discriminator" );
        }
        return m_u_.my_string;
    }
    const std::string& my_string() const  {
        if (_d() != (MyUnionDiscriminator::USE_STRING) ) {
            throw ::dds::core::PreconditionNotMetError(
                "::MyUnion::my_string not selected by the discriminator" );
        }
        return m_u_.my_string;
    }
    void my_string(const ::omg::types::string_view& value) {
        m_u_.my_string = value;
        m_d_= (MyUnionDiscriminator::USE_STRING);
    }
    void my_string(::omg::types::string_view&& value) {
```

```
        m_u_.my_string = std::move(value);
        m_d_= (MyUnionDiscriminator::USE_STRING);
    }
    ::Foo& my_foo()  {
        if (_d() != (MyUnionDiscriminator::USE_FOO) ) {
            throw ::dds::core::PreconditionNotMetError(
                "::MyUnion::my_foo not selected by the discriminator" );
        }
        return m_u_.my_foo;
    }
    const ::Foo& my_foo() const  {
        if (_d() != (MyUnionDiscriminator::USE_FOO) ) {
            throw ::dds::core::PreconditionNotMetError(
                "::MyUnion::my_foo not selected by the discriminator" );
        }
        return m_u_.my_foo;
    }
    void my_foo(const ::Foo& value) {
        m_u_.my_foo = value;
        m_d_= (MyUnionDiscriminator::USE_FOO);
    }
    void my_foo(::Foo&& value) {
        m_u_.my_foo = std::move(value);
        m_d_= (MyUnionDiscriminator::USE_FOO);
    }
    bool operator == (const MyUnion& other_) const;
    bool operator != (const MyUnion& other_) const;
    static ::MyUnionDiscriminator default_discriminator();
    void swap(MyUnion& other_) noexcept ;
  private:
    ::MyUnionDiscriminator m_d_ {};
    struct NDDSUSERDllExport Union_ {
        int32_t my_long {};
        std::string my_string {};
        ::Foo my_foo {};
        Union_();
        Union_(int32_t my_long_,const ::omg::types::string_view& my_string_,const ::Foo& my_foo_);
    };
    Union_ m_u_;
};
inline void swap(MyUnion& a, MyUnion& b)  noexcept
{
    a.swap(b);
}
NDDSUSERDllExport std::ostream& operator<<(std::ostream& o, const MyUnion& sample);
struct NDDSUSERDllExport MyOtherType {
    int32_t m1 {};
    double m2 {};
    std::string m3 {};
    MyOtherType();
    MyOtherType(int32_t m1_,double m2_,const ::omg::types::string_view& m3_);
};
NDDSUSERDllExport bool operator == (const MyOtherType& a, const MyOtherType& b);
NDDSUSERDllExport bool operator != (const MyOtherType& a, const MyOtherType& b);
NDDSUSERDllExport void swap(MyOtherType& a, MyOtherType& b)  noexcept;
NDDSUSERDllExport std::ostream& operator<<(std::ostream& o, const MyOtherType& sample);
#ifdef NDDS_STANDALONE_TYPE
namespace rti {
    namespace topic {
        template <>
        struct default_enumerator< ::Color>
        {
            static const ::Color value;
        };
        template <>
        struct default_enumerator< ::MyUnionDiscriminator>
        {
            static const ::MyUnionDiscriminator value;
        };
    }
}
#else
namespace rti {
    namespace flat {
        namespace topic {
        }
    }
}
namespace dds {
    namespace topic {
        template<>
```

```
struct topic_type_name< ::Foo > {
    NDDSUSERDllExport static std::string value() {
        return "Foo";
    }
};
template<>
struct is_topic_type< ::Foo > : public ::dds::core::true_type {};
template<>
struct topic_type_support< ::Foo > {
    NDDSUSERDllExport
    static void register_type(
        ::dds::domain::DomainParticipant& participant,
        const std::string & type_name);
    NDDSUSERDllExport
    static std::vector<char>& to_cdr_buffer(
        std::vector<char>& buffer,
        const ::Foo& sample,
        ::dds::core::policy::DataRepresentationId representation
        = ::dds::core::policy::DataRepresentation::auto_id());
    NDDSUSERDllExport
    static void from_cdr_buffer(::Foo& sample, const std::vector<char>& buffer);
    NDDSUSERDllExport
    static void reset_sample(::Foo& sample);
    NDDSUSERDllExport
    static void allocate_sample(::Foo& sample, int, int);
    static const ::rti::topic::TypePluginKind::type type_plugin_kind =
    ::rti::topic::TypePluginKind::STL;
};
template<>
struct topic_type_name< ::MyType > {
    NDDSUSERDllExport static std::string value() {
        return "MyType";
    }
};
template<>
struct is_topic_type< ::MyType > : public ::dds::core::true_type {};
template<>
struct topic_type_support< ::MyType > {
    NDDSUSERDllExport
    static void register_type(
        ::dds::domain::DomainParticipant& participant,
        const std::string & type_name);
    NDDSUSERDllExport
    static std::vector<char>& to_cdr_buffer(
        std::vector<char>& buffer,
        const ::MyType& sample,
        ::dds::core::policy::DataRepresentationId representation
        = ::dds::core::policy::DataRepresentation::auto_id());
    NDDSUSERDllExport
    static void from_cdr_buffer(::MyType& sample, const std::vector<char>& buffer);
    NDDSUSERDllExport
    static void reset_sample(::MyType& sample);
    NDDSUSERDllExport
    static void allocate_sample(::MyType& sample, int, int);
    static const ::rti::topic::TypePluginKind::type type_plugin_kind =
    ::rti::topic::TypePluginKind::STL;
};
template<>
struct topic_type_name< ::MyUnion > {
    NDDSUSERDllExport static std::string value() {
        return "MyUnion";
    }
};
template<>
struct is_topic_type< ::MyUnion > : public ::dds::core::true_type {};
template<>
struct topic_type_support< ::MyUnion > {
    NDDSUSERDllExport
    static void register_type(
        ::dds::domain::DomainParticipant& participant,
        const std::string & type_name);
    NDDSUSERDllExport
    static std::vector<char>& to_cdr_buffer(
        std::vector<char>& buffer,
        const ::MyUnion& sample,
        ::dds::core::policy::DataRepresentationId representation
        = ::dds::core::policy::DataRepresentation::auto_id());
    NDDSUSERDllExport
    static void from_cdr_buffer(::MyUnion& sample, const std::vector<char>& buffer);
    NDDSUSERDllExport
    static void reset_sample(::MyUnion& sample);
```

```
            NDDSUSERDllExport
            static void allocate_sample(::MyUnion& sample, int, int);
            static const ::rti::topic::TypePluginKind::type type_plugin_kind =
            ::rti::topic::TypePluginKind::STL;
        };
        template<>
        struct topic_type_name< ::MyOtherType > {
            NDDSUSERDllExport static std::string value() {
                return "MyOtherType";
            }
        };
        template<>
        struct is_topic_type< ::MyOtherType > : public ::dds::core::true_type {};
        template<>
        struct topic_type_support< ::MyOtherType > {
            NDDSUSERDllExport
            static void register_type(
                ::dds::domain::DomainParticipant& participant,
                const std::string & type_name);
            NDDSUSERDllExport
            static std::vector<char>& to_cdr_buffer(
                std::vector<char>& buffer,
                const ::MyOtherType& sample,
                ::dds::core::policy::DataRepresentationId representation
                = ::dds::core::policy::DataRepresentation::auto_id());
            NDDSUSERDllExport
            static void from_cdr_buffer(::MyOtherType& sample, const std::vector<char>& buffer);
            NDDSUSERDllExport
            static void reset_sample(::MyOtherType& sample);
            NDDSUSERDllExport
            static void allocate_sample(::MyOtherType& sample, int, int);
            static const ::rti::topic::TypePluginKind::type type_plugin_kind =
            ::rti::topic::TypePluginKind::STL;
        };
    }
}
namespace rti {
    namespace topic {
        template<>
        struct dynamic_type< ::Foo > {
            typedef ::dds::core::xtypes::StructType type;
            NDDSUSERDllExport static const ::dds::core::xtypes::StructType& get();
        };
        template <>
        struct extensibility< ::Foo > {
            static const ::dds::core::xtypes::ExtensibilityKind::type kind =
            ::dds::core::xtypes::ExtensibilityKind::EXTENSIBLE;    };
        template<>
        struct dynamic_type< ::MyType > {
            typedef ::dds::core::xtypes::StructType type;
            NDDSUSERDllExport static const ::dds::core::xtypes::StructType& get();
        };
        template <>
        struct extensibility< ::MyType > {
            static const ::dds::core::xtypes::ExtensibilityKind::type kind =
            ::dds::core::xtypes::ExtensibilityKind::EXTENSIBLE;    };
        template <>
        struct default_enumerator< ::Color>
        {
            static const ::Color value;
        };
        template<>
        struct dynamic_type< ::Color > {
            typedef ::dds::core::xtypes::EnumType type;
            NDDSUSERDllExport static const ::dds::core::xtypes::EnumType& get();
        };
        template <>
        struct extensibility< ::Color > {
            static const ::dds::core::xtypes::ExtensibilityKind::type kind =
            ::dds::core::xtypes::ExtensibilityKind::EXTENSIBLE;    };
        template <>
        struct default_enumerator< ::MyUnionDiscriminator>
        {
            static const ::MyUnionDiscriminator value;
        };
        template<>
        struct dynamic_type< ::MyUnionDiscriminator > {
            typedef ::dds::core::xtypes::EnumType type;
            NDDSUSERDllExport static const ::dds::core::xtypes::EnumType& get();
        };
        template <>
```

```
        struct extensibility< ::MyUnionDiscriminator > {
            static const ::dds::core::xtypes::ExtensibilityKind::type kind =
            ::dds::core::xtypes::ExtensibilityKind::EXTENSIBLE;    };
        template<>
        struct dynamic_type< ::MyUnion > {
            typedef ::dds::core::xtypes::UnionType type;
            NDDSUSERDllExport static const ::dds::core::xtypes::UnionType& get();
        };
        template <>
        struct extensibility< ::MyUnion > {
            static const ::dds::core::xtypes::ExtensibilityKind::type kind =
            ::dds::core::xtypes::ExtensibilityKind::EXTENSIBLE;    };
        template<>
        struct dynamic_type< ::MyOtherType > {
            typedef ::dds::core::xtypes::StructType type;
            NDDSUSERDllExport static const ::dds::core::xtypes::StructType& get();
        };
        template <>
        struct extensibility< ::MyOtherType > {
            static const ::dds::core::xtypes::ExtensibilityKind::type kind =
            ::dds::core::xtypes::ExtensibilityKind::EXTENSIBLE;    };
    }
}
#endif // NDDS_STANDALONE_TYPE
#if defined(NDDS_USER_DLL_EXPORT) || defined(NDDS_USER_SYMBOL_EXPORT)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport
#endif
#endif // Foo_982570024_hpp
```

9.3 Foo_publisher.cxx

The unmodified publication example generated by rtiddsgen using the C++11 option for the -language flag.

See also

Publication Example (p. 304)

```
/*
* (c) Copyright, Real-Time Innovations, 2020.  All rights reserved.
* RTI grants Licensee a license to use, modify, compile, and create derivative
* works of the software solely for use with RTI Connext DDS. Licensee may
* redistribute copies of the software provided that all such copies are subject
* to this license. The software is provided "as is", with no warranty of any
* type, including any warranty for fitness for any purpose. RTI is under no
* obligation to maintain or support the software. RTI shall not be liable for
* any incidental or consequential damages arising out of the use or inability
* to use the software.
*/
#include <iostream>
#include <dds/pub/ddspub.hpp>
#include <rti/util/util.hpp>      // for sleep()
#include <rti/config/Logger.hpp>  // for logging
// alternatively, to include all the standard APIs:
//  <dds/dds.hpp>
// or to include both the standard APIs and extensions:
//  <rti/rti.hpp>
//
// For more information about the headers and namespaces, see:
//
       https://community.rti.com/static/documentation/connext-dds/7.7.0/doc/api/connext_dds/api_cpp2/group__DDSNamespaceModule.htm
// For information on how to use extensions, see:
//
       https://community.rti.com/static/documentation/connext-dds/7.7.0/doc/api/connext_dds/api_cpp2/group__DDSCpp2Conventions.htm
#include "application.hpp"  // for command line parsing and ctrl-c
#include "Foo.hpp"
void run_publisher_application(unsigned int domain_id, unsigned int sample_count)
{
    // DDS objects behave like shared pointers or value types
    // (see https://community.rti.com/best-practices/use-modern-c-types-correctly)
    // Pulls XML QoS files from one of the Default locations such as
    // NDDS_QOS_PROFILES ENV var or ./NDDS_QOS_PROFILES.xml
    dds::core::QosProvider qos_provider = dds::core::QosProvider::Default();
```

```
    // Create DomainParticipant using QoS Profile from Library
    dds::domain::qos::DomainParticipantQos participant_qos =
    qos_provider.participant_qos("Foo_Library::Participant_Profile");
    // Start communicating in a domain, usually one participant per application
    dds::domain::DomainParticipant participant(domain_id, participant_qos);
    // Create a Topic with a name and a datatype
    dds::topic::Topic< ::MyOtherType> topic(participant, "Example MyOtherType");
    // Create a Publisher
    dds::pub::Publisher publisher(participant);
    dds::pub::qos::DataWriterQos writer_qos =
    qos_provider.datawriter_qos("Foo_Library::Periodic_Best_Effort_Profile");
    // Create a DataWriter with default QoS
    dds::pub::DataWriter< ::MyOtherType> writer(publisher, topic, writer_qos);
    ::MyOtherType data;
    // Main loop, write data
    for (unsigned int samples_written = 0;
    !application::shutdown_requested && samples_written < sample_count;
    samples_written++) {
        // Modify the data to be written here
        data.m1 = static_cast< int32_t>(samples_written);
        std::cout « "Writing ::MyOtherType, count " « samples_written « std::endl;
        writer.write(data);
        // Send once every second
        rti::util::sleep(dds::core::Duration(1));
    }
}
int main(int argc, char *argv[])
{
    using namespace application;
    // Parse arguments and handle control-C
    auto arguments = parse_arguments(argc, argv);
    if (arguments.parse_result == ParseReturn::exit) {
        return EXIT_SUCCESS;
    } else if (arguments.parse_result == ParseReturn::failure) {
        return EXIT_FAILURE;
    }
    setup_signal_handlers();
    // Sets Connext verbosity to help debugging
    rti::config::Logger::instance().verbosity(arguments.verbosity);
    try {
        run_publisher_application(arguments.domain_id, arguments.sample_count);
    } catch (const std::exception& ex) {
        // This will catch DDS exceptions
        std::cerr « "Exception in run_publisher_application(): " « ex.what()
        « std::endl;
        return EXIT_FAILURE;
    }
    // Releases the memory used by the participant factory.  Optional at
    // application exit
    dds::domain::DomainParticipant::finalize_participant_factory();
    return EXIT_SUCCESS;
}
```

9.4 Foo_subscriber.cxx

The unmodified subscription example generated by rtiddsgen using the C++11 option for the -language flag.

See also

Subscription Example (p. 305)

```
/*
* (c) Copyright, Real-Time Innovations, 2020.  All rights reserved.
* RTI grants Licensee a license to use, modify, compile, and create derivative
* works of the software solely for use with RTI Connext DDS. Licensee may
* redistribute copies of the software provided that all such copies are subject
* to this license. The software is provided "as is", with no warranty of any
* type, including any warranty for fitness for any purpose. RTI is under no
* obligation to maintain or support the software. RTI shall not be liable for
* any incidental or consequential damages arising out of the use or inability
* to use the software.
*/
#include <algorithm>
```

```
#include <iostream>
#include <dds/sub/ddssub.hpp>
#include <dds/core/ddscore.hpp>
#include <rti/config/Logger.hpp>  // for logging
// alternatively, to include all the standard APIs:
//  <dds/dds.hpp>
// or to include both the standard APIs and extensions:
//  <rti/rti.hpp>
//
// For more information about the headers and namespaces, see:
//
       https://community.rti.com/static/documentation/connext-dds/7.7.0/doc/api/connext_dds/api_cpp2/group__DDSNamespaceModule.htm
// For information on how to use extensions, see:
//
       https://community.rti.com/static/documentation/connext-dds/7.7.0/doc/api/connext_dds/api_cpp2/group__DDSCpp2Conventions.htm
#include "Foo.hpp"
#include "application.hpp"  // for command line parsing and ctrl-c
int process_data(dds::sub::DataReader< ::MyOtherType> reader)
{
    // Take all samples
    int count = 0;
    dds::sub::LoanedSamples< ::MyOtherType> samples = reader.take();
    for (auto sample : samples) {
        if (sample.info().valid()) {
            count++;
            std::cout « sample.data() « std::endl;
        } else {
            std::cout « "Instance state changed to "
            « sample.info().state().instance_state() « std::endl;
        }
    }
    return count;
} // The LoanedSamples destructor returns the loan
void run_subscriber_application(unsigned int domain_id, unsigned int sample_count)
{
    // DDS objects behave like shared pointers or value types
    // (see https://community.rti.com/best-practices/use-modern-c-types-correctly)
    // Pulls XML QoS files from one of the Default locations such as
    // NDDS_QOS_PROFILES ENV var or ./NDDS_QOS_PROFILES.xml
    dds::core::QosProvider qos_provider = dds::core::QosProvider::Default();
    // Create DomainParticipant using QoS Profile from Library
    dds::domain::qos::DomainParticipantQos participant_qos =
    qos_provider.participant_qos("Foo_Library::Participant_Profile");
    // Start communicating in a domain, usually one participant per application
    dds::domain::DomainParticipant participant(domain_id, participant_qos);
    // Create a Topic with a name and a datatype
    dds::topic::Topic< ::MyOtherType> topic(participant, "Example MyOtherType");
    // Create a Subscriber and DataReader with default Qos
    dds::sub::Subscriber subscriber(participant);
    // Create DataReader using QoS Profile from Library
    dds::sub::qos::DataReaderQos reader_qos =
    qos_provider.datareader_qos("Foo_Library::Periodic_Best_Effort_Profile");
    dds::sub::DataReader< ::MyOtherType> reader(subscriber, topic, reader_qos);
    // Create a ReadCondition for any data received on this reader and set a
    // handler to process the data
    unsigned int samples_read = 0;
    dds::sub::cond::ReadCondition read_condition(
        reader,
        dds::sub::status::DataState::any(),
        [reader, &samples_read]() { samples_read += process_data(reader); });
    // WaitSet will be woken when the attached condition is triggered
    dds::core::cond::WaitSet waitset;
    waitset += read_condition;
    while (!application::shutdown_requested && samples_read < sample_count) {
        std::cout « "::MyOtherType subscriber sleeping up to 1 sec..." « std::endl;
        // Run the handlers of the active conditions. Wait for up to 1 second.
        waitset.dispatch(dds::core::Duration(1));
    }
}
int main(int argc, char *argv[])
{
    using namespace application;
    // Parse arguments and handle control-C
    auto arguments = parse_arguments(argc, argv);
    if (arguments.parse_result == ParseReturn::exit) {
        return EXIT_SUCCESS;
    } else if (arguments.parse_result == ParseReturn::failure) {
        return EXIT_FAILURE;
    }
    setup_signal_handlers();
    // Sets Connext verbosity to help debugging
```

```
    rti::config::Logger::instance().verbosity(arguments.verbosity);
    try {
        run_subscriber_application(arguments.domain_id, arguments.sample_count);
    } catch (const std::exception& ex) {
        // This will catch DDS exceptions
        std::cerr « "Exception in run_subscriber_application(): " « ex.what()
        « std::endl;
        return EXIT_FAILURE;
    }
    // Releases the memory used by the participant factory.  Optional at
    // application exit
    dds::domain::DomainParticipant::finalize_participant_factory();
    return EXIT_SUCCESS;
}
```

9.5 USER_QOS_PROFILES.xml

The file that defines the QoS profiles that Foo_publisher.cxx and Foo_subscriber.cxx define.

See also

Configuring QoS Profiles with XML (p. 302)

set_qos (abstract) (p. 1522)

```
1 <?xml version="1.0"?>
2
3 <dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="http://community.rti.com/schema/current/rti_dds_profiles.xsd">
4
5     <qos_library name="RequestReplyExampleProfiles">
6
7         <!-- Use these profiles to customize the Requester or Replier QoS -->
8
9         <!-- These profiles inherit from the built-in profile
10             BuiltinQosLib::Pattern.RPC, which defines the default values for
11             the DataWriters and DataReaders created by a Requester or Replier
12         -->
13         <qos_profile name="RequesterExampleProfile" base_name="BuiltinQosLib::Pattern.RPC">
14             <!-- Set the QoS for the DataReader created by the Requester -->
15             <!--
16             <datareader_qos>
17                 ...
18             </datareader_qos>
19             -->
20
21             <!-- Set the QoS for the DataWriter created by the Requester -->
22             <!--
23             <datawriter_qos>
24                 ...
25             </datawriter_qos>
26             -->
27         </qos_profile>
28
29         <qos_profile name="ReplierExampleProfile" base_name="BuiltinQosLib::Pattern.RPC">
30
31             <!-- Set the QoS for the DataReader created by the Replier -->
32             <!--
33             <datareader_qos>
34                 ...
35             </datareader_qos>
36             -->
37
38             <!-- Set the QoS for the DataWriter created by the Replier -->
39             <!--
40             <datawriter_qos>
41                 ...
42             </datawriter_qos>
43             -->
44         </qos_profile>
45
46     </qos_library>
47 </dds>
```

Index

_d
MyFlatUnionOffset, 1782
~AbstractBuilder
rti::flat::AbstractBuilder, 783
~CoherentAccess
dds::sub::CoherentAccess, 931
~CoherentSet
dds::pub::CoherentSet, 933
~LoanedDynamicData
rti::core::xtypes::LoanedDynamicData, 1657
~LoanedSamples
dds::sub::LoanedSamples< T >, 1666
~RobotControl
rpc_example::RobotControl, 2214
~RobotControlAsync
rpc_example::RobotControlAsync, 2216
~ScopedLoggerVerbosity
rti::config::ScopedLoggerVerbosity, 2319
~SuspendedPublication
dds::pub::SuspendedPublication, 2442
~UnregisterThreadOnExit
rti::core::UnregisterThreadOnExit, 2590
~external
dds::core::external< T >, 1554
~optional
dds::core::optional< T >, 1883

absolute_generation
dds::sub::Rank, 2132
accept_unknown_discriminator
rti::config::compliance::XTypesMask, 2671
accept_unknown_enum_value
rti::config::compliance::XTypesMask, 2670
accept_unknown_peers
rti::core::policy::Discovery, 1270
access_scope
dds::core::policy::Presentation, 1947
acknowledge_all
dds::sub::DataReader< T >, 1013, 1014
rti::queuing::QueueConsumer< T >, 2069
rti::queuing::QueueReplier< TReq, TRep >, 2100
rti::queuing::QueueRequester< TReq, TRep >, 2121
acknowledge_reply
rti::queuing::QueueRequester< TReq, TRep >, 2121
acknowledge_request
rti::queuing::QueueReplier< TReq, TRep >, 2100
acknowledge_sample
dds::sub::DataReader< T >, 1015
rti::queuing::QueueConsumer< T >, 2069
acknowledgment_kind
dds::core::policy::Reliability, 2154
AcknowledgmentKind
RELIABILITY, 545
AckResponseData
rti::sub::AckResponseData, 799
active_count
rti::core::status::ReliableReaderActivityChangedStatus, 2158
ACTIVITY
rti::util::heap_monitoring::SnapshotContentFormat_def, 2371
Activity Context, 449
set_attribute_mask, 450
add_cookie
rti::topic::WriterContentFilterHelper< T, CompileData, WriterFilterData >, 2659
add_member
dds::core::xtypes::EnumType, 1538
dds::core::xtypes::StructType, 2408, 2409
dds::core::xtypes::UnionType, 2588, 2589
add_members
dds::core::xtypes::EnumType, 1538
dds::core::xtypes::StructType, 2409
dds::core::xtypes::UnionType, 2588
add_my_final
MyFlatMutableBuilder, 1768
MyFlatUnionBuilder, 1780
add_my_final_array
MyFlatMutableBuilder, 1768
add_my_optional_primitive
MyFlatMutableBuilder, 1767
add_my_primitive
MyFlatMutableBuilder, 1767
MyFlatUnionBuilder, 1779
add_my_primitive_array
MyFlatMutableBuilder, 1767
add_n
rti::flat::FinalSequenceBuilder< ElementOffset >, 1570
rti::flat::PrimitiveSequenceBuilder< T >, 1956, 1957
add_next

rti::flat::FinalSequenceBuilder< ElementOffset >, 1570
rti::flat::PrimitiveSequenceBuilder< T >, 1955
add_parameter
dds::sub::Query, 2058
dds::topic::Filter, 1561
add_peer
dds::domain::DomainParticipant, 1352
Additional RTI Connext Communication Patterns, 361
address
NDDS_Transport_Interface_t, 1785
rti::core::Locator, 1675
address_bit_count
NDDS_Transport_Property_t, 1788
addresses
rti::core::policy::MulticastMapping, 1748
ADMIN_CONSOLE
rti::core::policy::ServiceKind_def, 2358
AdminConsoleService
rti::core::policy::Service, 2353
advance
rti::flat::SequenceIterator< E, OffsetKind >, 2333
AGGREGATION_TYPE
dds::core::xtypes::TypeKind_def, 2572
alert
Logging, 458
rti::config::Logger, 1689, 1691
ALIAS_TYPE
dds::core::xtypes::TypeKind_def, 2573
AliasType
dds::core::xtypes::AliasType, 801
ALIVE
rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_def, 1251
alive
dds::sub::status::InstanceState, 1617
alive_count
dds::core::status::LivelinessChangedStatus, 1653
alive_count_change
dds::core::status::LivelinessChangedStatus, 1653
alive_instance_count
rti::core::status::DataReaderCacheStatus, 1051
rti::core::status::DataWriterCacheStatus, 1200
alive_instance_count_peak
rti::core::status::DataReaderCacheStatus, 1051
rti::core::status::DataWriterCacheStatus, 1201
alive_instance_removal
rti::core::DataReaderResourceLimitsInstanceReplacementSettings, 1110
ALIVE_OR_DISPOSED
rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_def, 1251
ALIVE_THEN_DISPOSED
rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_def, 1251
All
rti::core::policy::TransportBuiltin, 2535
all
dds::core::status::StatusMask, 2379
DISCOVERY_CONFIG, 527
rti::config::activity_context::AttributeKindMask, 867
rti::core::CompressionIdMask, 941
rti::core::policy::TransportBuiltinMask, 2539
rti::util::network_capture::ContentKindMask, 964
rti::util::network_capture::TrafficKindMask, 2532
WIRE_PROTOCOL, 556
All QoS Policies, 508
QosPolicyCountSeq, 514
QosPolicyId, 514
all_categories
rti::config::LogCategory_def, 1683
AllocationSettings
rti::core::AllocationSettings, 804
allocator_type
rti::core::bounded_sequence< T, MaxLength >, 891
allow_interfaces_list
NDDS_Transport_Property_t, 1789
allow_interfaces_list_length
NDDS_Transport_Property_t, 1790
allow_multicast_interfaces_list
NDDS_Transport_Property_t, 1791
allow_multicast_interfaces_list_length
NDDS_Transport_Property_t, 1792
ALLOW_TYPE_COERCION
dds::core::policy::TypeConsistencyEnforcementKind_def, 2571
AllowTypeCoercion
dds::core::policy::TypeConsistencyEnforcement, 2569
any
dds::sub::status::DataState, 1124
dds::sub::status::InstanceState, 1618
dds::sub::status::SampleState, 2318
dds::sub::status::ViewState, 2616
rti::sub::status::DataStateEx, 1131
rti::sub::status::StreamKind, 2392
any_data
dds::sub::status::DataState, 1125
rti::sub::status::DataStateEx, 1132
ANY_INSTANCE
rti::core::policy::DataReaderInstanceRemovalKind_def, 1055
AnyDataReader
dds::sub::AnyDataReader, 808
AnyDataWriter
dds::pub::AnyDataWriter, 816
AnyTopic

dds::topic::AnyTopic, 826
api
rti::config::LogCategory_def, 1683
app_ack_period
rti::core::RtpsReliableReaderProtocol, 2227
append_to_expression_parameter
dds::topic::ContentFilteredTopic< T >, 958
Application Acknowledgement, 158
Application Startup, 161
Application-Defined Metadata, 159
APPLICATION_AUTO
rti::core::policy::AcknowledgmentKind_def, 798
APPLICATION_EXPLICIT
rti::core::policy::AcknowledgmentKind_def, 798
application_name
rti::core::policy::Monitoring, 1709, 1710
APPLICATION_ORDERED
rti::core::policy::AcknowledgmentKind_def, 798
Applied Per-Instance, 287
ARRAY_TYPE
dds::core::xtypes::TypeKind_def, 2573
ArrayType
dds::core::xtypes::ArrayType, 829–831
assert_liveliness
dds::domain::DomainParticipant, 1338
dds::pub::DataWriter< T >, 1172
assertions_per_lease_duration
dds::core::policy::Liveliness, 1651
async_waitset_property
dds::rpc::ServerParams, 2349, 2350
ASYNCHRONOUS
rti::core::policy::PublishModeKind_def, 2019
Asynchronous
rti::core::policy::PublishMode, 2015, 2016
asynchronous_batch_thread
rti::core::policy::AsynchronousPublisher, 837, 838
ASYNCHRONOUS_PUBLISHER, 515
asynchronous_publisher
rti::core::policy::DiscoveryConfig, 1299, 1300
AsynchronousPublisher
rti::core::policy::AsynchronousPublisher, 835
AsyncWaitSet, 507
Ignore, 508
rti::core::cond::AsyncWaitSet, 844, 845
AsyncWaitSetCompletionToken
rti::core::cond::AsyncWaitSetCompletionToken, 855
AsyncWaitSetProperty
rti::core::cond::AsyncWaitSetProperty, 859
at
dds::core::vector< T >, 2608
rti::core::bounded_sequence< T, MaxLength >, 897
attach_condition
dds::core::cond::WaitSet, 2624
rti::core::cond::AsyncWaitSet, 848
attach_reader
rti::sub::SampleProcessor, 2309
AttributeKindMask
rti::config::activity_context::AttributeKindMask, 864
AUTO
rti::core::policy::CdrPaddingKind_def, 921
AUTO_COUNT
rti::core::AllocationSettings, 805
auto_id
DATA_REPRESENTATION, 521
auto_max_total_instances
DATA_READER_RESOURCE_LIMITS, 519
AUTO_TYPE_COERCION
dds::core::policy::TypeConsistencyEnforcementKind_def, 2571
auto_writer_depth
DURABILITY, 531
autodispose_unregistered_instances
dds::core::policy::WriterDataLifecycle, 2661, 2662
AutoDisposeUnregisteredInstances
dds::core::policy::WriterDataLifecycle, 2662
AutoEnable
dds::core::policy::EntityFactory, 1529
autoenable_created_entities
dds::core::policy::EntityFactory, 1529
AUTOMATIC
dds::core::policy::LivelinessKind_def, 1655
rti::core::policy::TransportMulticastKind_def, 2547
Automatic
dds::core::policy::Liveliness, 1650
automatic
dds::core::Duration, 1451
rti::core::Guid, 1597
rti::core::SampleIdentity, 2284
rti::core::SequenceNumber, 2338
autopurge_disposed_instances_delay
dds::core::policy::ReaderDataLifecycle, 2142
dds::core::policy::WriterDataLifecycle, 2664
autopurge_disposed_samples_delay
dds::core::policy::ReaderDataLifecycle, 2141
autopurge_nowriter_instances_delay
dds::core::policy::ReaderDataLifecycle, 2143
autopurge_nowriter_samples_delay
dds::core::policy::ReaderDataLifecycle, 2140
autopurge_remote_not_alive_writer_delay
rti::core::policy::DataReaderResourceLimits, 1103, 1104
autopurge_remote_virtual_writer_delay
rti::core::policy::DataReaderResourceLimits, 1104, 1105
autopurge_unregistered_instances_delay
dds::core::policy::WriterDataLifecycle, 2662, 2663
AutoPurgeDisposedSamples
dds::core::policy::ReaderDataLifecycle, 2141

AutoPurgeNoWriterSamples
dds::core::policy::ReaderDataLifecycle, 2142
autoregister_instances
rti::core::policy::DataWriterResourceLimits, 1242, 1243
AutoTypeCoercion
dds::core::policy::TypeConsistencyEnforcement, 2570
AVAILABILITY, 516
EndpointGroupSeq, 516
Availability
rti::core::policy::Availability, 870
available_builtin_endpoints_ext
dds::topic::ParticipantBuiltinTopicData, 1912

back
rti::core::bounded_sequence< T, MaxLength >, 898
BackwardsCompatible
WIRE_PROTOCOL, 558
banish_ignored_participants
dds::domain::DomainParticipant, 1364
rti::domain, 728
baseline
Builtin Qos Profiles, 469
baseline_5_0_0
Builtin Qos Profiles, 469
baseline_5_1_0
Builtin Qos Profiles, 470
baseline_5_2_0
Builtin Qos Profiles, 470
baseline_5_3_0
Builtin Qos Profiles, 470
baseline_6_0_0
Builtin Qos Profiles, 470
baseline_6_1_0
Builtin Qos Profiles, 470
baseline_7_0_0
Builtin Qos Profiles, 470
baseline_7_1_0
Builtin Qos Profiles, 471
Basic QoS, 104
basic_string
dds::core::basic_string< CharType, Allocator >, 876, 877
BATCH, 517
Batch
rti::core::policy::Batch, 881
Batching, 161
begin
dds::core::policy::GroupData, 1593
dds::core::policy::TopicData, 2502
dds::core::policy::UserData, 2594
dds::core::vector< T >, 2610
dds::sub, 679
dds::sub::LoanedSamples< T >, 1667–1669
dds::sub::Query, 2056, 2057
dds::sub::SharedSamples< T, DELEGATE >, 2364
dds::topic::Filter, 1560, 1561
rti::core::bounded_sequence< T, MaxLength >, 899
rti::flat::AbstractAlignedList< ElementOffset >, 782
rti::sub, 765
rti::sub::AckResponseData, 799
rti::sub::ValidLoanedSamples< T >, 2597–2599
BEST_EFFORT
dds::core::policy::ReliabilityKind_def, 2157
BestEffort
dds::core::policy::Reliability, 2154
bind_and_manage_listener
rti::core, 709
rti::core::ListenerBinder< Entity, Listener >, 1644, 1645
bind_listener
rti::core, 708
rti::core::ListenerBinder< Entity, Listener >, 1644
binding_ping_period
NDDS_Transport_UDPv4_WAN_Property_t, 1818
bitmask
rti::topic::trust::EndpointTrustProtectionInfo, 1519
rti::topic::trust::ParticipantTrustProtectionInfo, 1921
BOOLEAN_TYPE
dds::core::xtypes::TypeKind_def, 2572
bounded_sequence
rti::core::bounded_sequence< T, MaxLength >, 893–895
Supporting Types and Constants, 436
bounds
dds::core::xtypes::UnidimensionalCollectionType, 2577
buffer_alignment
rti::core::policy::ReceiverPool, 2148
buffer_initial_size
rti::core::xtypes::DynamicDataProperty, 1498
buffer_max_size
rti::core::xtypes::DynamicDataProperty, 1499
buffer_size
rti::core::policy::ReceiverPool, 2147
build_data
FlatData Builders, 411
build_my_final_seq
MyFlatMutableBuilder, 1768
build_my_mutable
MyFlatMutableBuilder, 1769
MyFlatUnionBuilder, 1779
build_my_mutable_array
MyFlatMutableBuilder, 1769
build_my_mutable_seq
MyFlatMutableBuilder, 1769
build_my_primitive_seq

MyFlatMutableBuilder, 1768
build_my_string
MyFlatMutableBuilder, 1769
build_my_string_seq
MyFlatMutableBuilder, 1769
build_next
rti::flat::MutableArrayBuilder< ElementBuilder, N >, 1754
rti::flat::MutableSequenceBuilder< ElementBuilder >, 1758
build_version
rti::config::LibraryVersion, 1635
Built-in Topic's Trust Types, 54
Built-in Topics, 46
Built-in Transport Plugins, 83
Built-in Types, 50
Built-in Types Examples, 601
Builtin Qos Profiles, 462
baseline, 469
baseline_5_0_0, 469
baseline_5_1_0, 470
baseline_5_2_0, 470
baseline_5_3_0, 470
baseline_6_0_0, 470
baseline_6_1_0, 470
baseline_7_0_0, 470
baseline_7_1_0, 471
generic_510_transport_compatibility, 472
generic_auto_tuning, 478, 488
generic_best_effort, 474, 483
generic_common, 471
generic_connext_micro_compatibility, 471
generic_connext_micro_compatibility_2_4_3, 472
generic_connext_micro_compatibility_2_4_9, 472
generic_keep_last_reliable, 473, 483
generic_keep_last_reliable_large_data, 476, 485
generic_keep_last_reliable_large_data_fast_flow, 477, 486
generic_keep_last_reliable_large_data_medium_flow, 477, 487
generic_keep_last_reliable_large_data_slow_flow, 477, 487
generic_keep_last_reliable_persistent, 478, 488
generic_keep_last_reliable_transient, 478, 487
generic_keep_last_reliable_transient_local, 477, 487
generic_minimal_memory_footprint, 478, 488
generic_monitoring2, 479
generic_monitoring2_wan, 479
generic_monitoring_common, 471
generic_other_dds_vendor_compatibility, 472
generic_participant_large_data, 474, 484
generic_participant_large_data_monitoring, 475, 484
generic_security, 473
generic_strict_reliable, 473, 483
generic_strict_reliable_high_throughput, 474, 483
generic_strict_reliable_large_data, 475, 485
generic_strict_reliable_large_data_fast_flow, 476, 486
generic_strict_reliable_large_data_medium_flow, 476, 486
generic_strict_reliable_large_data_slow_flow, 476, 486
generic_strict_reliable_low_latency, 474, 484
library_name, 469, 482, 491
pattern_alarm_event, 481, 490
pattern_alarm_status, 482, 490
pattern_event, 481, 489
pattern_last_value_cache, 482, 491
pattern_periodic_data, 480, 488
pattern_reliable_streaming, 481, 489
pattern_rpc, 480
pattern_status, 482, 490
pattern_streaming, 480, 489
snippet_compatibility_5_1_0_transport_enable, 506
snippet_compatibility_connext_micro_version_2_4_3, 506
snippet_compatibility_other_dds_vendors_enable, 506
snippet_feature_auto_tuning_enable, 500
snippet_feature_monitoring2_enable, 501
snippet_feature_monitoring_enable, 501
snippet_feature_security_enable, 502
snippet_feature_topic_query_enable, 502
snippet_feature_type_support_v1_disable, 503
snippet_feature_type_support_v1_enable, 502
snippet_feature_type_support_v2_disable, 503
snippet_feature_type_support_v2_enable, 503
snippet_optimization_discovery_common, 494
snippet_optimization_discovery_endpoint_fast, 495
snippet_optimization_discovery_participant_compact, 495
snippet_optimization_reliability_protocol_common, 491
snippet_optimization_reliability_protocol_dynamicmemalloc, 494
snippet_optimization_reliability_protocol_high_rate, 492
snippet_optimization_reliability_protocol_keep_all, 491
snippet_optimization_reliability_protocol_keep_last, 492
snippet_optimization_reliability_protocol_large_data, 493
snippet_optimization_reliability_protocol_low_latency, 493
snippet_optimization_transport_large_buffers, 495
snippet_qos_policy_batching_enable, 499
snippet_qos_policy_durability_persistent, 498

snippet_qos_policy_durability_transient, 498
snippet_qos_policy_durability_transient_local, 498
snippet_qos_policy_flow_controller_209mbps, 499
snippet_qos_policy_flow_controller_52mbps, 500
snippet_qos_policy_flow_controller_838mbps, 499
snippet_qos_policy_history_keep_all, 497
snippet_qos_policy_history_keep_last_1, 497
snippet_qos_policy_publish_mode_asynchronous, 497
snippet_qos_policy_reliability_best_effort, 496
snippet_qos_policy_reliability_reliable, 496
snippet_transport_tcp_lan_client, 504
snippet_transport_tcp_wan_asymmetric_client, 505
snippet_transport_tcp_wan_asymmetric_server, 504
snippet_transport_tcp_wan_symmetric_client, 504
snippet_transport_udp_avoid_ip_fragmentation, 505
snippet_transport_udp_wan, 505
builtin_discovery_plugins
rti::core::policy::DiscoveryConfig, 1293, 1294
builtin_endpoints_required_mask
rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo, 1918
builtin_kx_endpoints_required_mask
rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo, 1919
builtin_multicast
rti::core::policy::RtpsReservedPortKindMask, 2258
builtin_multicast_port_offset
rti::core::RtpsWellKnownPorts, 2263, 2264
builtin_subscriber
dds::sub, 671
dds::sub::Subscriber, 2419
builtin_unicast
rti::core::policy::RtpsReservedPortKindMask, 2257
builtin_unicast_port_offset
rti::core::RtpsWellKnownPorts, 2264
BuiltinTopicReaderResourceLimits
rti::core::policy::BuiltinTopicReaderResourceLimits, 908
BY_RECEPTION_TIMESTAMP
dds::core::policy::DestinationOrderKind_def, 1264
BY_SOURCE_TIMESTAMP
dds::core::policy::DestinationOrderKind_def, 1264
bytes_per_token
rti::pub::FlowControllerTokenBucketProperty, 1587, 1588
ByteSeq
Supporting Types and Constants, 435
BytesTopicType
dds::core::BytesTopicType, 918
bzip2
rti::core::CompressionIdMask, 942

c_api_version
Version, 461
c_build_number
Version, 461
c_str
dds::core::basic_string< CharType, Allocator >, 877
can_convert
dds::core::xtypes::DynamicData, 1492
rti::core::xtypes, 723
cancel_asynchronous
rti::core::ThreadSettingsKindMask, 2456
capacity
dds::core::vector< T >, 2607
rti::core::bounded_sequence< T, MaxLength >, 902
rti::flat::AbstractBuilder, 784
cbegin
rti::core::bounded_sequence< T, MaxLength >, 899
cdr_padding_kind
rti::core::policy::TypeSupport, 2576
cdr_serialized_sample_key_max_size
dds::core::xtypes::AbstractConstructedType< MemberType >, 790
cdr_serialized_sample_max_size
dds::core::xtypes::AbstractConstructedType< MemberType >, 788
cdr_serialized_sample_min_size
dds::core::xtypes::AbstractConstructedType< MemberType >, 789
CdrPaddingKind
TYPESUPPORT, 554
cend
rti::core::bounded_sequence< T, MaxLength >, 900
change
rti::core::status::EventCount< IntegerType >, 1545
channel_filter_expression_max_length
rti::core::policy::DomainParticipantResourceLimits, 1424
channel_seq_max_length
rti::core::policy::DomainParticipantResourceLimits, 1423
channels
rti::core::policy::MultiChannel, 1751, 1752
ChannelSettings
rti::core::ChannelSettings, 922
ChannelSettingsSeq
MULTICHANNEL, 538
CHAR_8_TYPE
dds::core::xtypes::TypeKind_def, 2573
check_crc
rti::core::policy::WireProtocol, 2640
checkpoint_thread_settings
rti::util::network_capture::NetworkCaptureParams, 1833
class_id
rti::core::TransportInfo, 2542, 2543

classid
NDDS_Transport_Property_t, 1787
cleanup_period
rti::core::policy::Database, 972, 973
clear
dds::core::vector< T >, 2607
rti::core::bounded_sequence< T, MaxLength >, 902
clear_all_members
dds::core::xtypes::DynamicData, 1479
clear_member
dds::core::xtypes::DynamicData, 1480, 1481
clear_optional_member
dds::core::xtypes::DynamicData, 1480
Client-side API, 407
RobotControlClientEndpoint, 407
ClientEndpoint
dds::rpc::ClientEndpoint< Request, Reply >, 926
ClientParams
dds::rpc::ClientParams, 928
Clock Selection, 43
clone
rti::flat::Sample< OffsetType >, 2277
close
dds::core::Entity, 1525
dds::pub::AnyDataWriter, 818
dds::rpc::ClientEndpoint< Request, Reply >, 926
dds::rpc::Server, 2348
dds::rpc::ServiceEndpoint< Dispatcher >, 2356
dds::sub::AnyDataReader, 811
dds::sub::cond::ReadCondition, 2136
dds::sub::DataReader< T >, 1024
dds::topic::AnyTopic, 827
rti::pub::FlowController, 1576
rti::sub::TopicQuery, 2518
close_contained_entities
dds::domain::DomainParticipant, 1345
dds::pub::Publisher, 2002
dds::sub::DataReader< T >, 1016
dds::sub::Subscriber, 2418
closed
dds::rpc::ClientEndpoint< Request, Reply >, 926
dds::sub::cond::ReadCondition, 2136
rti::pub::FlowController, 1576
rti::sub::TopicQuery, 2518
coherent_access
dds::core::policy::Presentation, 1947, 1948
coherent_set_info
dds::sub::SampleInfo, 2295
coherent_set_sequence_number
rti::core::CoherentSetInfo, 936, 937
CoherentAccess
dds::sub::CoherentAccess, 930
CoherentSet
dds::pub::CoherentSet, 932
CoherentSetInfo
rti::core::CoherentSetInfo, 935
COLLECTION_TYPE
dds::core::xtypes::TypeKind_def, 2572
collector_initial_peers
rti::core::MonitoringDedicatedParticipantSettings, 1715, 1716
comm_ports_list
NDDS_Transport_UDPv4_WAN_Property_t, 1816
comm_ports_list_length
NDDS_Transport_UDPv4_WAN_Property_t, 1817
communication
rti::config::LogCategory_def, 1683
compare
dds::core::Duration, 1453
dds::core::Time, 2462
compile
rti::topic::ContentFilter< T, CompileData >, 952
Compliance Configuration, 451
get_rpc_mask, 453
get_xtypes_mask, 451
load_compliance_masks, 453
set_rpc_mask, 454
set_xtypes_mask, 452
COMPRESSED
rti::util::heap_monitoring::SnapshotOutputFormat_def, 2372
compressed_sample_count
rti::core::status::DataReaderCacheStatus, 1053
compression_ids
rti::core::CompressionSettings, 945
compression_level_best_compression
rti::core::CompressionSettings, 945
compression_level_best_speed
rti::core::CompressionSettings, 945
compression_level_default
rti::core::CompressionSettings, 945
compression_settings
dds::core::policy::DataRepresentation, 1117
CompressionIdMask
rti::core::CompressionIdMask, 941
CompressionSettings
rti::core::CompressionSettings, 944
compute_crc
rti::core::policy::WireProtocol, 2640
Concurrency, 164
concurrency_level
rti::core::MonitoringEventDistributionSettings, 1723, 1724
rti::core::MonitoringLoggingDistributionSettings, 1729
condition
dds::sub, 663
dds::sub::DataReader< T >, 1026

dds::sub::DataReader< T >::Selector, 2324
conditions
dds::core::cond::WaitSet, 2626
rti::core::cond::AsyncWaitSet, 853
Conditions and WaitSets, 426
ConditionSeq
dds::core::cond::WaitSet, 2621
Configuring QoS Profiles with XML, 302
const_iterator
dds::sub::cond::QueryCondition, 2061
rti::core::bounded_sequence< T, MaxLength >, 892
rti::sub::ValidLoanedSamples< T >, 2596
const_pointer
rti::core::bounded_sequence< T, MaxLength >, 892
const_reference
rti::core::bounded_sequence< T, MaxLength >, 892
const_reverse_iterator
rti::core::bounded_sequence< T, MaxLength >, 893
ConstOffset
MyFlatFinalOffset, 1760
MyFlatMutableOffset, 1771
MyFlatUnionOffset, 1781
rti::flat::Sample< OffsetType >, 2275
CONSTRUCTED_TYPE
dds::core::xtypes::TypeKind_def, 2572
consumer
rti::queuing::QueueReplier< TReq, TRep >, 2101
rti::queuing::QueueRequester< TReq, TRep >, 2122
contains_entity
dds::domain::DomainParticipant, 1339
content
dds::sub, 663
dds::sub::DataReader< T >, 1025
dds::sub::DataReader< T >::Selector, 2324
Content Filtering, 165
content_filter_allocation
rti::core::policy::DomainParticipantResourceLimits, 1404
content_filter_dropped_sample_count
rti::core::status::DataReaderCacheStatus, 1049
content_filter_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1412
content_filter_property
dds::topic::SubscriptionBuiltinTopicData, 2436
content_filter_topic_name
rti::core::ContentFilterProperty, 961
content_filtered_topic_allocation
rti::core::policy::DomainParticipantResourceLimits, 1403
content_filtered_topic_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1412
content_type
dds::core::xtypes::CollectionType, 939
contentfilter_property_max_length
rti::core::policy::DomainParticipantResourceLimits, 1422, 1423
ContentFilteredTopic
dds::topic::ContentFilteredTopic< T >, 955
ContentKindMask
rti::util::network_capture::ContentKindMask, 963
CONTINUOUS
rti::sub::TopicQuerySelectionKind_def, 2529
Conventions, 350
convert
dds::core::xtypes::DynamicData, 1491, 1492
rti::core::xtypes, 720, 722, 723
Cookie
rti::core::Cookie, 966
cookie
rti::pub::WriteParams, 2647
cookie_max_length
rti::core::policy::DataWriterResourceLimits, 1241
CookieSeq
rti::core::Cookie, 967
copy_to_sample
rti::sub, 765
rti::sub::LoanedSample< T >, 1663
core_build_number
Version, 461
core_version
Version, 460
corrupted_rtps_message_count
rti::core::status::DomainParticipantProtocolStatus, 1383
corrupted_rtps_message_count_change
rti::core::status::DomainParticipantProtocolStatus, 1383
count
dds::core::policy::QosPolicyCount, 2021
cpp_api_version
Version, 461
cpp_build_number
Version, 461
cpu_list
rti::core::ThreadSettings, 2451
cpu_rotation
rti::core::ThreadSettings, 2452
crbegin
rti::core::bounded_sequence< T, MaxLength >, 900
create_data
dds::pub::DataWriter< T >, 1182
rti::flat::Sample< OffsetType >, 2276
create_empty
dds::core::policy::DataRepresentation, 1115
create_participant_from_config
dds::core::QosProvider, 2046

create_qos_provider_ex
dds::core::QosProvider, 2047
create_query_condition_ex
Query Conditions, 66
create_read_condition_ex
dds::sub::cond::ReadCondition, 2136, 2137
create_topic_query_data_from_service_request
rti::sub, 768
rti::sub::TopicQueryData, 2521
create_type_from_tuple
dds::core::xtypes::StructType, 2410
rti::core::xtypes, 725
Creating Custom Content Filters, 333
Creating New Transport Plugins, 84
crend
rti::core::bounded_sequence< T, MaxLength >, 901
critical
Logging, 458
rti::config::Logger, 1689, 1691
current
rti::core::ProductVersion, 1969
rti::core::ProtocolVersion, 1977
current_count
dds::core::status::PublicationMatchedStatus, 1991
dds::core::status::SubscriptionMatchedStatus, 2440
rti::core::status::ServiceRequestAcceptedStatus, 2360
current_count_change
dds::core::status::PublicationMatchedStatus, 1992
dds::core::status::SubscriptionMatchedStatus, 2440
current_count_peak
dds::core::status::PublicationMatchedStatus, 1992
dds::core::status::SubscriptionMatchedStatus, 2441
current_time
dds::domain::DomainParticipant, 1340
Custom Content Filters, 571
find_content_filter, 572
no_compile_data, 572
CustomFilter
rti::topic::CustomFilter< T >, 968

data
dds::core::BytesTopicType, 919
dds::core::StringTopicType, 2398, 2399
dds::sub::Sample< T >, 2272
rti::core::bounded_sequence< T, MaxLength >, 898, 899
rti::sub::LoanedSample< T >, 1661
Data Availability, 167
Data Samples, 66
Data State, 71
Data Writers, 56
data_available
dds::core::status::StatusMask, 2383
data_on_readers
dds::core::status::StatusMask, 2382
data_reader
dds::sub::cond::ReadCondition, 2136
dds::sub::Query, 2059
DATA_READER_PROTOCOL, 517
DATA_READER_RESOURCE_LIMITS, 518
auto_max_total_instances, 519
DataReaderInstanceRemovalKind, 518
DATA_REPRESENTATION, 519
auto_id, 521
DataRepresentationId, 520
DataRepresentationIdSeq, 520
xcdr, 520
xcdr2, 521
xml, 520
data_state
rti::sub::status::DataStateEx, 1129
DATA_TAG, 523
data_tag
dds::topic::PublicationBuiltinTopicData, 1985
dds::topic::SubscriptionBuiltinTopicData, 2434
DATA_WRITER_PROTOCOL, 521
DATA_WRITER_RESOURCE_LIMITS, 522
DataWriterResourceLimitsInstanceReplacementKind, 522
DATA_WRITER_TRANSFER_MODE, 523
DATABASE, 517
database
rti::config::LogCategory_def, 1683
DATABASE_INTEGRATION
rti::core::policy::ServiceKind_def, 2358
DatabaseIntegrationService
rti::core::policy::Service, 2353
DataReader
dds::sub::DataReader< T >, 983–989, 991
datareader
rti::sub::TopicQuery, 2518
DataReader Use Cases, 316
datareader_cache
dds::core::status::StatusMask, 2386
datareader_cache_status
dds::sub::DataReader< T >, 1012
datareader_protocol
dds::core::status::StatusMask, 2387
datareader_protocol_status
dds::sub::DataReader< T >, 1013
datareader_qos
dds::core::QosProvider, 2033
rti::request::ReplierParams, 2182
rti::request::RequesterParams, 2201
datareader_qos_w_topic_name
dds::core::QosProvider, 2039
DataReaderInstanceRemovalKind

DATA_READER_RESOURCE_LIMITS, 518
DataReaderProtocol
rti::core::policy::DataReaderProtocol, 1060
DataReaderQos
dds::sub::qos::DataReaderQos, 1075, 1076
DataReaderResourceLimits
rti::core::policy::DataReaderResourceLimits, 1085
DataReaderResourceLimitsInstanceReplacementSettings
rti::core::DataReaderResourceLimitsInstanceReplacementSettings, 1109
DataReaders, 63
DataReaderStatusConditionHandler
rti::sub::cond::DataReaderStatusConditionHandler< T >, 1112
DataRepresentation
dds::core::policy::DataRepresentation, 1115
DataRepresentationId
DATA_REPRESENTATION, 520
DataRepresentationIdSeq
DATA_REPRESENTATION, 520
DataState
dds::sub::status::DataState, 1120, 1121
DataStateEx
rti::sub::status::DataStateEx, 1128
DataTag
dds::core::policy::DataTag, 1134, 1135
DataType
rti::sub::LoanedSample< T >, 1661
DataWriter
dds::pub::DataWriter< T >, 1144–1148
DataWriter Use Cases, 311
datawriter_application_acknowledgment
dds::core::status::StatusMask, 2387
datawriter_cache
dds::core::status::StatusMask, 2386
datawriter_cache_status
dds::pub::DataWriter< T >, 1176
datawriter_instance_replaced
dds::core::status::StatusMask, 2387
datawriter_protocol
dds::core::status::StatusMask, 2386
datawriter_protocol_status
dds::pub::DataWriter< T >, 1176
datawriter_qos
dds::core::QosProvider, 2035
rti::request::ReplierParams, 2182
rti::request::RequesterParams, 2201
datawriter_qos_profile_name
rti::core::MonitoringEventDistributionSettings, 1724, 1725
rti::core::MonitoringLoggingDistributionSettings, 1730, 1731
rti::core::MonitoringPeriodicDistributionSettings, 1742, 1743
datawriter_qos_w_topic_name
dds::core::QosProvider, 2040
DataWriterProtocol
rti::core::policy::DataWriterProtocol, 1211
DataWriterQos
dds::pub::qos::DataWriterQos, 1229, 1230
DataWriterResourceLimits
rti::core::policy::DataWriterResourceLimits, 1237
DataWriterResourceLimitsInstanceReplacementKind
DATA_WRITER_RESOURCE_LIMITS, 522
DataWriterShmemRefTransferModeSettings
rti::core::DataWriterShmemRefTransferModeSettings, 1252
DataWriterTransferMode
rti::core::policy::DataWriterTransferMode, 1254
dds, 615
DDS Extension QoS, 116
DDS Specification QoS, 112
dds::all, 615
dds::core, 616
InstanceHandleSeq, 619
operator!=, 624–626
operator<<, 622, 626
operator*, 620, 621
operator+, 622, 623
operator-, 623, 624
operator/, 621
operator==, 624, 625
polymorphic_cast, 620
swap, 622
dds::core::AlreadyClosedError, 806
what, 806
dds::core::basic_string< CharType, Allocator >, 875
basic_string, 876, 877
c_str, 877
operator std::basic_string< CharType >, 879
operator!=, 878
operator<<, 879
operator=, 878
operator==, 878
size, 877
to_std_string, 879
dds::core::BytesTopicType, 917
BytesTopicType, 918
data, 919
length, 920
operator std::vector< uint8_t >, 919
operator[], 920
dds::core::cond, 627
dds::core::cond::Condition, 948
dispatch, 949
trigger_value, 949
dds::core::cond::GuardCondition, 1594
handler, 1594

reset_handler, 1595
trigger_value, 1595
dds::core::cond::StatusCondition, 2372
enabled_statuses, 2373, 2374
entity, 2374
handler, 2374
reset_handler, 2375
StatusCondition, 2373
dds::core::cond::WaitSet, 2617
attach_condition, 2624
conditions, 2626
ConditionSeq, 2621
detach_all, 2626
detach_condition, 2625
dispatch, 2623, 2624
operator+=, 2624
operator-=, 2624
property, 2627
wait, 2621–2623
wait_one, 2626
WaitSet, 2621
WaitSetImpl, 2621
dds::core::Duration, 1448
automatic, 1451
compare, 1453
Duration, 1450
from_microsecs, 1451
from_millisecs, 1451
from_secs, 1452
infinite, 1451
nanosec, 1453
operator!=, 1455
operator<, 1456
operator<=, 1455
operator>, 1454
operator>=, 1454
operator*, 1459
operator+, 1457
operator+=, 1456
operator-, 1457
operator-=, 1456
operator/, 1460
operator==, 1454
sec, 1452
swap, 1460
to_chrono, 1458
to_microsecs, 1458
to_millisecs, 1458
to_nanosecs, 1457
to_secs, 1458
zero, 1450
dds::core::Entity, 1520
close, 1525
enable, 1524
instance_handle, 1525
polymorphic_cast, 1526
retain, 1526
status_changes, 1525
dds::core::Error, 1539
Error, 1540
what, 1540
dds::core::Exception, 1546
what, 1547
dds::core::external< T >, 1551
~external, 1554
external, 1553, 1554
get, 1555, 1556
get_shared_ptr, 1556
is_locked, 1557
operator bool, 1557
operator!=, 1557
operator*, 1555
operator->, 1556, 1557
operator=, 1555
operator==, 1557
swap, 1558
dds::core::IllegalOperationError, 1608
what, 1608
dds::core::ImmutablePolicyError, 1609
what, 1609
dds::core::InconsistentPolicyError, 1610
what, 1610
dds::core::InstanceHandle, 1612
InstanceHandle, 1613
is_nil, 1614
nil, 1614
operator<<, 1614
operator==, 1613
dds::core::InvalidArgumentError, 1619
what, 1619
dds::core::InvalidDowncastError, 1620
what, 1620
dds::core::KeyedBytesTopicType, 1626
key, 1627
KeyedBytesTopicType, 1626
length, 1628
operator std::vector< uint8_t >, 1627
operator[], 1628
value, 1627, 1628
dds::core::KeyedStringTopicType, 1629
key, 1630
KeyedStringTopicType, 1629
value, 1630, 1631
dds::core::NotAllowedBySecurityError, 1868
what, 1868
dds::core::NotEnabledError, 1869
what, 1869
dds::core::NullReferenceError, 1870

what, 1870
dds::core::optional< T >, 1879
~optional, 1883
get, 1886
get_ptr, 1887
has_value, 1884
is_set, 1884
operator bool, 1885
operator!=, 1890, 1891
operator<<, 1891
operator*, 1887
operator->, 1887, 1888
operator=, 1888, 1889
operator==, 1889, 1890
optional, 1882, 1883
reset, 1885
set, 1884
swap, 1889
value, 1885, 1886
dds::core::OutOfResourcesError, 1898
what, 1899
dds::core::policy, 627
dds::core::policy::DataRepresentation, 1113
compression_settings, 1117
create_empty, 1115
DataRepresentation, 1115
value, 1116
dds::core::policy::DataTag, 1132
DataTag, 1134, 1135
Entry, 1134
exists, 1135
get, 1135
get_all, 1137
remove, 1137
set, 1135, 1136
size, 1137
try_get, 1136
dds::core::policy::Deadline, 1255
Deadline, 1257
period, 1258
dds::core::policy::DestinationOrder, 1258
DestinationOrder, 1260, 1261
kind, 1261
ReceptionTimestamp, 1262
scope, 1262
source_timestamp_tolerance, 1262, 1263
SourceTimestamp, 1261
dds::core::policy::DestinationOrderKind_def, 1263
BY_RECEPTION_TIMESTAMP, 1264
BY_SOURCE_TIMESTAMP, 1264
type, 1264
dds::core::policy::Durability, 1433
direct_communication, 1438, 1439
Durability, 1436, 1437
kind, 1437
Persistent, 1438
storage_settings, 1440, 1441
Transient, 1438
TransientLocal, 1438
Volatile, 1438
writer_depth, 1439, 1440
dds::core::policy::DurabilityKind_def, 1441
PERSISTENT, 1442
TRANSIENT, 1442
TRANSIENT_LOCAL, 1442
type, 1442
VOLATILE, 1442
dds::core::policy::DurabilityService, 1443
DurabilityService, 1444
history_depth, 1446
history_kind, 1445
max_instances, 1447
max_samples, 1446
max_samples_per_instance, 1447
service_cleanup_delay, 1445
dds::core::policy::EntityFactory, 1527
AutoEnable, 1529
autoenable_created_entities, 1529
ManuallyEnable, 1529
TEntityFactory, 1528, 1529
dds::core::policy::GroupData, 1591
begin, 1593
end, 1593
GroupData, 1592
value, 1593
dds::core::policy::History, 1602
depth, 1605
History, 1604
KeepAll, 1605
KeepLast, 1606
kind, 1604
dds::core::policy::HistoryKind_def, 1606
KEEP_ALL, 1607
KEEP_LAST, 1607
type, 1606
dds::core::policy::LatencyBudget, 1631
duration, 1633
LatencyBudget, 1633
dds::core::policy::Lifespan, 1635
duration, 1637
Lifespan, 1636
dds::core::policy::Liveliness, 1645
assertions_per_lease_duration, 1651
Automatic, 1650
kind, 1649
lease_duration, 1649, 1650
Liveliness, 1648
ManualByParticipant, 1650

ManualByTopic, 1651
dds::core::policy::LivelinessKind_def, 1654
AUTOMATIC, 1655
MANUAL_BY_PARTICIPANT, 1655
MANUAL_BY_TOPIC, 1655
type, 1654
dds::core::policy::Ownership, 1899
Exclusive, 1905
kind, 1904, 1905
Ownership, 1904
Shared, 1905
dds::core::policy::OwnershipKind_def, 1905
EXCLUSIVE, 1906
SHARED, 1906
type, 1906
dds::core::policy::OwnershipStrength, 1906
OwnershipStrength, 1907
value, 1908
dds::core::policy::Partition, 1923
name, 1926
TPartition, 1925, 1926
dds::core::policy::policy_id< Policy >, 1940
dds::core::policy::policy_name< Policy >, 1941
dds::core::policy::Presentation, 1942
access_scope, 1947
coherent_access, 1947, 1948
drop_incomplete_coherent_set, 1949, 1950
GroupAccessScope, 1949
InstanceAccessScope, 1949
ordered_access, 1948
Presentation, 1946
TopicAccessScope, 1949
dds::core::policy::PresentationAccessScopeKind_def, 1950
GROUP, 1951
HIGHEST_OFFERED, 1951
INSTANCE, 1951
TOPIC, 1951
type, 1951
dds::core::policy::QosPolicyCount, 2020
count, 2021
policy_id, 2021
QosPolicyCount, 2020
dds::core::policy::ReaderDataLifecycle, 2137
autopurge_disposed_instances_delay, 2142
autopurge_disposed_samples_delay, 2141
autopurge_nowriter_instances_delay, 2143
autopurge_nowriter_samples_delay, 2140
AutoPurgeDisposedSamples, 2141
AutoPurgeNoWriterSamples, 2142
NoAutoPurge, 2141
ReaderDataLifecycle, 2139, 2140
dds::core::policy::Reliability, 2149
acknowledgment_kind, 2154
BestEffort, 2154
exact_reliability_kind_match, 2155, 2156
instance_state_consistency_kind, 2155
kind, 2153
max_blocking_time, 2153
Reliability, 2152
Reliable, 2154
dds::core::policy::ReliabilityKind_def, 2156
BEST_EFFORT, 2157
RELIABLE, 2157
type, 2157
dds::core::policy::ResourceLimits, 2203
initial_instances, 2208
initial_samples, 2207
instance_hash_buckets, 2208, 2209
max_instances, 2206
max_samples, 2205, 2206
max_samples_per_instance, 2207
ResourceLimits, 2205
dds::core::policy::TimeBasedFilter, 2469
minimum_separation, 2472
TimeBasedFilter, 2471, 2472
dds::core::policy::TopicData, 2500
begin, 2502
end, 2503
TopicData, 2501, 2502
value, 2502
dds::core::policy::TransportPriority, 2552
TransportPriority, 2553
value, 2553, 2554
dds::core::policy::TypeConsistencyEnforcement, 2563
AllowTypeCoercion, 2569
AutoTypeCoercion, 2570
DisallowTypeCoercion, 2569
force_type_validation, 2568
ignore_enum_literal_names, 2569
ignore_member_names, 2567
ignore_sequence_bounds, 2566
ignore_string_bounds, 2567
kind, 2565, 2566
prevent_type_widening, 2568
TypeConsistencyEnforcement, 2565
dds::core::policy::TypeConsistencyEnforcementKind_def, 2570
ALLOW_TYPE_COERCION, 2571
AUTO_TYPE_COERCION, 2571
DISALLOW_TYPE_COERCION, 2571
type, 2570
dds::core::policy::UserData, 2591
begin, 2594
end, 2594
UserData, 2592, 2593
value, 2593, 2594
dds::core::policy::WriterDataLifecycle, 2660

autodispose_unregistered_instances, 2661, 2662
AutoDisposeUnregisteredInstances, 2662
autopurge_disposed_instances_delay, 2664
autopurge_unregistered_instances_delay, 2662, 2663
ManuallyDisposeUnregisteredInstances, 2662
WriterDataLifecycle, 2661
dds::core::PreconditionNotMetError, 1941
what, 1942
dds::core::QosProvider, 2025
create_participant_from_config, 2046
create_qos_provider_ex, 2047
datareader_qos, 2033
datareader_qos_w_topic_name, 2039
datawriter_qos, 2035
datawriter_qos_w_topic_name, 2040
Default, 2036
default_library, 2041, 2042
default_profile, 2041, 2042
default_profile_library, 2042
default_provider_params, 2036, 2037
default_qos_provider_params, 2048
load_profiles, 2045
participant_factory_qos, 2037
participant_qos, 2031
profiles_loaded, 2043
provider_params, 2044, 2045
publisher_qos, 2034
qos_profile_libraries, 2043
qos_profiles, 2043
QosProvider, 2030
reload_profiles, 2045
reset_default, 2036
subscriber_qos, 2032, 2033
topic_qos, 2032
topic_qos_w_topic_name, 2038
type, 2044
unload_profiles, 2045
USE_DDS_DEFAULT_QOS_PROFILE, 2047
dds::core::Reference< DELEGATE >, 2149
dds::core::safe_enum< def, inner >, 2265
inner_enum, 2267
operator!=, 2268
operator<, 2268
operator<<, 2270
operator<=, 2269
operator>, 2269
operator>=, 2269
operator==, 2268
safe_enum, 2267
swap, 2269
underlying, 2268
dds::core::status, 630
get_status, 631
dds::core::status::InconsistentTopicStatus, 1611
total_count, 1612
total_count_change, 1612
dds::core::status::LivelinessChangedStatus, 1652
alive_count, 1653
alive_count_change, 1653
last_publication_handle, 1654
not_alive_count, 1653
not_alive_count_change, 1653
dds::core::status::LivelinessLostStatus, 1655
total_count, 1656
total_count_change, 1656
dds::core::status::OfferedDeadlineMissedStatus, 1871
last_instance_handle, 1872
total_count, 1872
total_count_change, 1872
dds::core::status::OfferedIncompatibleQosStatus, 1873
last_policy_id, 1874
policies, 1874
total_count, 1873
total_count_change, 1874
dds::core::status::PublicationMatchedStatus, 1990
current_count, 1991
current_count_change, 1992
current_count_peak, 1992
last_subscription_handle, 1992
total_count, 1991
total_count_change, 1991
dds::core::status::RequestedDeadlineMissedStatus, 2184
last_instance_handle, 2185
total_count, 2184
total_count_change, 2184
dds::core::status::RequestedIncompatibleQosStatus, 2185
last_policy_id, 2186
policies, 2186
total_count, 2186
total_count_change, 2186
dds::core::status::SampleLostStatus, 2303
last_reason, 2304
total_count, 2304
total_count_change, 2304
dds::core::status::SampleRejectedState, 2310
not_rejected, 2312
rejected_by_decode_failure, 2314
rejected_by_instances_limit, 2313
rejected_by_remote_writers_per_virtual_queue_limit, 2314
rejected_by_samples_limit, 2312
rejected_by_samples_per_instance_limit, 2313
rejected_by_samples_per_remote_writer_limit, 2313
SampleRejectedState, 2312
dds::core::status::SampleRejectedStatus, 2315
last_instance_handle, 2316

last_reason, 2315
total_count, 2315
total_count_change, 2315
dds::core::status::StatusMask, 2375
all, 2379
data_available, 2383
data_on_readers, 2382
datareader_cache, 2386
datareader_protocol, 2387
datawriter_application_acknowledgment, 2387
datawriter_cache, 2386
datawriter_instance_replaced, 2387
datawriter_protocol, 2386
get_status, 2389
inconsistent_topic, 2379
invalid_local_advance_notice, 2388
liveliness_changed, 2383
liveliness_lost, 2383
MaskType, 2378
none, 2379
offered_deadline_missed, 2379
offered_incompatible_qos, 2380
publication_matched, 2384
reliable_reader_activity_changed, 2385
reliable_writer_cache_changed, 2385
requested_deadline_missed, 2380
requested_incompatible_qos, 2381
sample_lost, 2381
sample_rejected, 2382
sample_removed, 2389
service_request_accepted, 2388
StatusMask, 2378
subscription_matched, 2384
dds::core::status::SubscriptionMatchedStatus, 2439
current_count, 2440
current_count_change, 2440
current_count_peak, 2441
last_publication_handle, 2440
total_count, 2440
total_count_change, 2440
dds::core::StringTopicType, 2395
data, 2398, 2399
operator const dds::core::string &, 2398
operator dds::core::string &, 2398
operator std::string, 2397
StringTopicType, 2396, 2397
dds::core::TEntityQos< DELEGATE >, 2446
operator<<, 2447
operator>>, 2448
operator=, 2448
policy, 2447
dds::core::Time, 2457
compare, 2462
from_microsecs, 2459
from_millisecs, 2460
from_secs, 2460
invalid, 2459
maximum, 2459
nanosec, 2461
operator!=, 2464
operator<, 2465
operator<=, 2464
operator>, 2462
operator>=, 2462
operator+, 2467, 2468
operator+=, 2465
operator-, 2468, 2469
operator-=, 2466
operator==, 2464
sec, 2460, 2461
Time, 2458
to_microsecs, 2466
to_millisecs, 2466
to_nanosecs, 2467
to_secs, 2467
zero, 2459
dds::core::TimeoutError, 2473
what, 2473
dds::core::UnsupportedError, 2590
what, 2590
dds::core::Value< D >, 2601
delegate, 2601, 2602
extensions, 2602
operator const D &, 2603
operator D&, 2602
operator->, 2601
dds::core::vector< T >, 2603
at, 2608
begin, 2610
capacity, 2607
clear, 2607
end, 2610
operator std::vector< T >, 2606
operator!=, 2610
operator<<, 2611
operator=, 2609
operator==, 2609
operator[], 2609
reserve, 2608
resize, 2607
size, 2607
swap, 2611
vector, 2605, 2606
dds::core::WeakReference< T >, 2630
dds::core::xtypes, 631
ExtensibilityKind, 633
is_aggregation_type, 634
is_collection_type, 633

is_constructed_type, 633
is_primitive_type, 633
operator<<, 634
primitive_type, 634
dds::core::xtypes::AbstractConstructedType< MemberType >, 785
cdr_serialized_sample_key_max_size, 790
cdr_serialized_sample_max_size, 788
cdr_serialized_sample_min_size, 789
extensibility_kind, 787
find_member_by_name, 788
INVALID_INDEX, 790
Member, 786
member, 787
member_count, 787
MemberIndex, 786
members, 788
dds::core::xtypes::AliasType, 801
AliasType, 801
is_pointer, 802
related_type, 802
resolve_alias, 802
unit, 802
dds::core::xtypes::ArrayType, 828
ArrayType, 829–831
dimension, 832
dimension_count, 832
total_element_count, 832
dds::core::xtypes::CollectionType, 939
content_type, 939
dds::core::xtypes::DynamicData, 1461
can_convert, 1492
clear_all_members, 1479
clear_member, 1480, 1481
clear_optional_member, 1480
convert, 1491, 1492
discriminator_value, 1477
DynamicData, 1467, 1468
from_cdr_buffer, 1490
from_string, 1489
get_cdr_buffer, 1481
get_discriminator, 1478
get_tuple, 1493
get_values, 1471–1473
info, 1484
is_cdr, 1481
is_member_key, 1488
loan_value, 1475–1477
member_count, 1483
member_exists, 1483, 1484
member_exists_in_type, 1484
member_index, 1487
member_info, 1484, 1486
member_type, 1489
set_cdr_buffer, 1482
set_discriminator, 1478
set_tuple, 1493
set_values, 1474
to_cdr_buffer, 1490
type, 1482
type_kind, 1483
value, 1468–1470
dds::core::xtypes::DynamicType, 1504
is_aggregation_type, 1507
is_assignable_from, 1506
is_collection_type, 1507
is_constructed_type, 1507
is_primitive_type, 1507
kind, 1506
name, 1506
operator<<, 1507
print_idl, 1508
to_string, 1508
dds::core::xtypes::EnumMember, 1533
EnumMember, 1533
name, 1534
ordinal, 1534
dds::core::xtypes::EnumType, 1535
add_member, 1538
add_members, 1538
EnumType, 1536, 1537
find_member_by_ordinal, 1538
dds::core::xtypes::ExtensibilityKind_def, 1550
EXTENSIBLE, 1551
FINAL, 1551
MUTABLE, 1551
type, 1551
dds::core::xtypes::Member, 1700
get_bitbound, 1704
get_id, 1703
has_bitbound, 1704
has_id, 1703
id, 1705
INVALID_ID, 1706
is_bitset, 1704
is_key, 1703
is_must_understand, 1704
is_optional, 1703
is_pointer, 1703
key, 1704
Member, 1702
must_understand, 1705
name, 1702, 1704
optional, 1705
pointer, 1705
type, 1703
dds::core::xtypes::PrimitiveType, 1959
primitive_type, 1960

dds::core::xtypes::SequenceType, 2344
SequenceType, 2345, 2346
dds::core::xtypes::StringType, 2399
StringType, 2400
dds::core::xtypes::StructType, 2400
add_member, 2408, 2409
add_members, 2409
create_type_from_tuple, 2410
extensibility_kind, 2408
find_member_by_id, 2408
has_parent, 2408
parent, 2408
StructType, 2402–2404, 2406, 2407
dds::core::xtypes::TypeKind_def, 2571
AGGREGATION_TYPE, 2572
ALIAS_TYPE, 2573
ARRAY_TYPE, 2573
BOOLEAN_TYPE, 2572
CHAR_8_TYPE, 2573
COLLECTION_TYPE, 2572
CONSTRUCTED_TYPE, 2572
ENUMERATION_TYPE, 2573
FLOAT_128_TYPE, 2573
FLOAT_32_TYPE, 2573
FLOAT_64_TYPE, 2573
INT_16_TYPE, 2572
INT_32_TYPE, 2573
INT_64_TYPE, 2573
INT_8_TYPE, 2573
OCTET_TYPE, 2573
PRIMITIVE_TYPE, 2572
SEQUENCE_TYPE, 2573
STRING_TYPE, 2574
STRUCTURE_TYPE, 2574
type, 2572
UINT_16_TYPE, 2572
UINT_32_TYPE, 2573
UINT_64_TYPE, 2573
UINT_8_TYPE, 2572
UNION_TYPE, 2574
WSTRING_TYPE, 2574
dds::core::xtypes::UnidimensionalCollectionType, 2576
bounds, 2577
dds::core::xtypes::UnionMember, 2578
DEFAULT_LABEL, 2583
DiscriminatorType, 2579
get_id, 2581
has_id, 2581
id, 2583
INVALID_ID, 2583
is_pointer, 2582
label, 2582
label_count, 2582
labels, 2582
LabelSeq, 2579
name, 2581, 2583
pointer, 2583
type, 2581
UnionMember, 2580
dds::core::xtypes::UnionType, 2584
add_member, 2588, 2589
add_members, 2588
discriminator, 2587
DiscriminatorType, 2585
find_member_by_id, 2587
find_member_by_label, 2587
UnionType, 2585–2587
dds::core::xtypes::WStringType, 2665
WStringType, 2665
dds::domain, 634
discovered_participant_data, 638
discovered_participants, 637, 638
find, 639
ignore, 635, 636
dds::domain::DomainParticipant, 1326
add_peer, 1352
assert_liveliness, 1338
banish_ignored_participants, 1364
close_contained_entities, 1345
contains_entity, 1339
current_time, 1340
default_datareader_qos, 1348
default_datawriter_qos, 1346, 1347
default_participant_qos, 1341
default_publisher_qos, 1342
default_subscriber_qos, 1343
default_topic_qos, 1344, 1345
delete_durable_subscription, 1357
discovered_participant_data, 1362
discovered_participant_subject_name, 1365
discovered_participants, 1361, 1362
discovered_participants_from_subject_name, 1366
dns_tracker_polling_period, 1355
domain_id, 1338
DomainParticipant, 1332, 1333
finalize_participant_factory, 1341
find, 1363
find_participant_by_name, 1367
find_participants, 1367, 1368
find_type, 1369
get_listener, 1336
ignore, 1359, 1360
is_type_registered, 1351
Listener, 1331
listener, 1334
operator<<, 1337
operator>>, 1337
participant_factory_qos, 1340

participant_protocol_status, 1358
property, 1338
qos, 1336, 1337
register_contentfilter, 1349
register_durable_subscription, 1358
register_type, 1370
remove_peer, 1353
resume_endpoint_discovery, 1356
set_listener, 1335, 1336
time_until_license_expiration, 1359
unregister_contentfilter, 1350
unregister_type, 1351
dds::domain::DomainParticipantListener, 1382
on_invalid_local_status_advance_notice, 1382
dds::domain::NoOpDomainParticipantListener, 1845
on_application_acknowledgment, 1849
on_data_available, 1851
on_data_on_readers, 1850
on_inconsistent_topic, 1852
on_instance_replaced, 1849
on_invalid_local_status_advance_notice, 1849
on_liveliness_changed, 1851
on_liveliness_lost, 1847
on_offered_deadline_missed, 1847
on_offered_incompatible_qos, 1847
on_publication_matched, 1848
on_reliable_reader_activity_changed, 1848
on_reliable_writer_cache_changed, 1848
on_requested_deadline_missed, 1850
on_requested_incompatible_qos, 1850
on_sample_lost, 1851
on_sample_rejected, 1850
on_sample_removed, 1848
on_service_request_accepted, 1849
on_subscription_matched, 1851
dds::domain::qos, 640
operator<<, 643, 645
to_string, 641–644
dds::domain::qos::DomainParticipantFactoryQos, 1378
operator<<, 1381
policy, 1379
to_string, 1379–1381
dds::domain::qos::DomainParticipantQos, 1384
DomainParticipantQos, 1386
operator<<, 1388, 1390
operator>>, 1388
policy, 1387
to_string, 1388–1390
dds::pub, 645
find, 651, 652
get, 647
ignore, 647, 648
matched_subscription_data, 650
matched_subscriptions, 648, 649
dds::pub::AnyDataWriter, 815
AnyDataWriter, 816
close, 818
get, 819, 820
operator=, 817, 819
publisher, 818
qos, 817
retain, 819
topic_name, 817
type_name, 818
wait_for_acknowledgments, 818
dds::pub::AnyDataWriterListener, 821
on_application_acknowledgment, 824
on_instance_replaced, 824
on_liveliness_lost, 822
on_offered_deadline_missed, 822
on_offered_incompatible_qos, 822
on_publication_matched, 823
on_reliable_reader_activity_changed, 823
on_reliable_writer_cache_changed, 823
on_sample_removed, 823
on_service_request_accepted, 824
dds::pub::CoherentSet, 932
∼CoherentSet, 933
CoherentSet, 932
end, 933
dds::pub::DataWriter< T >, 1138
assert_liveliness, 1172
create_data, 1182
DataWriter, 1144–1148
datawriter_cache_status, 1176
datawriter_protocol_status, 1176
delete_data, 1183
discard_loan, 1185
dispose_instance, 1162, 1163, 1173
find, 1192, 1193
find_datawriter_by_name, 1197, 1198
find_datawriter_by_topic_name, 1196
find_datawriters, 1194, 1195
flush, 1178
get_listener, 1170
get_loan, 1183
ignore, 1186, 1187
is_matched_subscription_active, 1191
is_sample_app_acknowledged, 1173
key_value, 1164, 1165
listener, 1169
liveliness_lost_status, 1171
lookup_instance, 1165
matched_subscription_data, 1188, 1192
matched_subscription_datawriter_protocol_status, 1177
matched_subscription_participant_data, 1190
matched_subscriptions, 1187, 1188

matched_subscriptions_locators, 1191
offered_deadline_missed_status, 1171
offered_incompatible_qos_status, 1171
operator<<, 1157, 1158, 1167
operator>>, 1167
publication_matched_status, 1172
publisher, 1168
qos, 1166
register_instance, 1158, 1159, 1185
reliable_reader_activity_changed_status, 1175
reliable_writer_cache_changed_status, 1175
service_request_accepted_status, 1178
set_listener, 1169, 1170
topic, 1168
unregister_instance, 1160, 1161, 1173
wait_for_acknowledgments, 1168
wait_for_asynchronous_publishing, 1174
write, 1148, 1149, 1153–1157, 1179
write_noexcept, 1180–1182
dds::pub::DataWriterListener< T >, 1202
on_application_acknowledgment, 1207
on_instance_replaced, 1206
on_liveliness_lost, 1204
on_offered_deadline_missed, 1203
on_offered_incompatible_qos, 1204
on_publication_matched, 1205
on_reliable_reader_activity_changed, 1206
on_reliable_writer_cache_changed, 1205
on_sample_removed, 1209
on_service_request_accepted, 1209
dds::pub::NoOpDataWriterListener< T >, 1841
on_application_acknowledgment, 1843
on_data_request, 1844
on_data_return, 1844
on_destination_unreachable, 1844
on_instance_replaced, 1843
on_liveliness_lost, 1842
on_offered_deadline_missed, 1842
on_offered_incompatible_qos, 1842
on_publication_matched, 1842
on_reliable_reader_activity_changed, 1843
on_reliable_writer_cache_changed, 1843
on_sample_removed, 1845
on_service_request_accepted, 1844
dds::pub::NoOpPublisherListener, 1852
on_application_acknowledgment, 1855
on_instance_replaced, 1855
on_liveliness_lost, 1854
on_offered_deadline_missed, 1853
on_offered_incompatible_qos, 1854
on_publication_matched, 1854
on_reliable_reader_activity_changed, 1855
on_reliable_writer_cache_changed, 1854
on_sample_removed, 1855
on_service_request_accepted, 1856
dds::pub::Publisher, 1993
close_contained_entities, 2002
default_datawriter_qos, 1998
find_publisher, 2004
find_publishers, 2003
get_listener, 2000
implicit_publisher, 2005
listener, 1999
operator<<, 1998
operator>>, 1998
participant, 2001
Publisher, 1995, 1996
qos, 1997
set_listener, 1999, 2000
wait_for_acknowledgments, 2001
wait_for_asynchronous_publishing, 2001
dds::pub::PublisherListener, 2006
dds::pub::qos, 653
operator<<, 656, 658
swap, 656
to_string, 654–657
dds::pub::qos::DataWriterQos, 1226
DataWriterQos, 1229, 1230
operator<<, 1232, 1234
operator>>, 1232
operator=, 1230
policy, 1231
to_string, 1233, 1234
dds::pub::qos::PublisherQos, 2007
operator<<, 2009, 2013
operator>>, 2010
policy, 2008, 2009
swap, 2010
to_string, 2011, 2012
dds::pub::SuspendedPublication, 2441
~SuspendedPublication, 2442
resume, 2443
SuspendedPublication, 2442
dds::rpc::ClientEndpoint< Request, Reply >, 925
ClientEndpoint, 926
close, 926
closed, 926
reply_datareader, 927
ReplyType, 926
request_datawriter, 927
RequestType, 926
wait_for_service, 927
dds::rpc::ClientParams, 928
ClientParams, 928
function_call_max_wait, 929
dds::rpc::RemoteInvalidArgumentError, 2162
what, 2162
dds::rpc::RemoteOutOfResourcesError, 2163

what, 2163
dds::rpc::RemoteUnknownExceptionError, 2165
what, 2166
dds::rpc::RemoteUnknownOperationError, 2166
what, 2167
dds::rpc::RemoteUnsupportedOperationError, 2167
what, 2168
dds::rpc::Server, 2346
close, 2348
run, 2347
Server, 2347
dds::rpc::ServerParams, 2348
async_waitset_property, 2349, 2350
thread_pool_size, 2349
dds::rpc::ServiceEndpoint< Dispatcher >, 2354
close, 2356
InterfaceType, 2355
reply_datawriter, 2357
ReplyType, 2356
request_datareader, 2357
RequestType, 2355
ServiceEndpoint, 2356
dds::sub, 658
begin, 679
builtin_subscriber, 671
condition, 663
content, 663
end, 679
find, 672, 673, 675, 677
get, 661
ignore, 667, 668
instance, 665
matched_publication_data, 670
matched_publications, 668, 669
max_samples, 662
move, 678
next_instance, 665
read, 661
state, 664
swap, 680
take, 662
dds::sub::AnyDataReader, 807
AnyDataReader, 808
close, 811
get, 810, 811
operator=, 809, 810
qos, 809
retain, 811
subscriber, 810
topic_name, 809
type_name, 810
dds::sub::AnyDataReaderListener, 812
on_data_available, 814
on_liveliness_changed, 814
on_requested_deadline_missed, 813
on_requested_incompatible_qos, 813
on_sample_lost, 814
on_sample_rejected, 813
on_subscription_matched, 814
dds::sub::CoherentAccess, 929
~CoherentAccess, 931
CoherentAccess, 930
end, 931
dds::sub::cond, 680
dds::sub::cond::QueryCondition, 2059
const_iterator, 2061
expression, 2062
iterator, 2060
parameters, 2062, 2063
parameters_length, 2063
QueryCondition, 2061
dds::sub::cond::ReadCondition, 2133
close, 2136
closed, 2136
create_read_condition_ex, 2136, 2137
data_reader, 2136
ReadCondition, 2134
state_filter, 2135
dds::sub::DataReader< T >, 976
acknowledge_all, 1013, 1014
acknowledge_sample, 1015
close, 1024
close_contained_entities, 1016
condition, 1026
content, 1025
DataReader, 983–989, 991
datareader_cache_status, 1012
datareader_protocol_status, 1013
default_filter_state, 991
find, 1037–1040
find_datareader_by_name, 1043, 1045
find_datareader_by_topic_description, 1044
find_datareader_by_topic_name, 1042
find_datareaders, 1040, 1041
get_listener, 1007
ignore, 1030, 1031
instance, 1028
is_data_consistent, 1022
is_matched_publication_alive, 1035
key_value, 1003, 1004
listener, 1006
liveliness_changed_status, 1011
lookup_instance, 1005
matched_publication_data, 1033, 1036
matched_publication_datareader_protocol_status, 1013
matched_publication_participant_data, 1034
matched_publications, 1031, 1032

max_samples, 1025
next_instance, 1028
operator<<, 1008
operator>>, 993, 995, 1009
qos, 1007, 1008
read, 996, 1000, 1001, 1018, 1019, 1024
read_noexcept, 1018
requested_deadline_missed_status, 1011
requested_incompatible_qos_status, 1012
sample_lost_status, 1011
sample_rejected_status, 1011
select, 1003
set_listener, 1006, 1007
state, 1027
subscriber, 1005
subscription_matched_status, 1012
take, 997, 1001, 1002, 1020, 1021, 1024
take_noexcept, 1018
topic_description, 1005
topic_name, 1023
type_name, 1023
wait_for_historical_data, 1009
dds::sub::DataReader< T >::ManipulatorSelector, 1697
operator>>, 1699
dds::sub::DataReader< T >::Selector, 2320
condition, 2324
content, 2324
instance, 2322
max_samples, 2325
next_instance, 2322
read, 2325–2327
reader, 2326
Selector, 2321
state, 2323
take, 2326–2328
dds::sub::DataReaderListener< T >, 1055
on_data_available, 1058
on_liveliness_changed, 1057
on_requested_deadline_missed, 1057
on_requested_incompatible_qos, 1057
on_sample_lost, 1058
on_sample_rejected, 1057
on_subscription_matched, 1058
dds::sub::GenerationCount, 1589
disposed, 1590
GenerationCount, 1589
no_writers, 1590
dds::sub::LoanedSamples< T >, 1663
~LoanedSamples, 1666
begin, 1667–1669
end, 1668, 1670
iterator, 1665
length, 1667
LoanedSamples, 1666
move, 1669
operator[], 1666
return_loan, 1667
return_loan_noexcept, 1667
swap, 1668, 1670
valid_data, 1671, 1672
dds::sub::NoOpDataReaderListener< T >, 1838
on_data_available, 1840
on_liveliness_changed, 1839
on_requested_deadline_missed, 1839
on_requested_incompatible_qos, 1839
on_sample_lost, 1840
on_sample_rejected, 1839
on_subscription_matched, 1840
dds::sub::NoOpSubscriberListener, 1864
on_data_available, 1866
on_data_on_readers, 1865
on_liveliness_changed, 1866
on_requested_deadline_missed, 1865
on_requested_incompatible_qos, 1865
on_sample_lost, 1866
on_sample_rejected, 1865
on_subscription_matched, 1866
dds::sub::qos, 681
operator<<, 684, 686
to_string, 681, 683–686
dds::sub::qos::DataReaderQos, 1072
DataReaderQos, 1075, 1076
operator<<, 1078, 1080
operator>>, 1078
operator=, 1076
policy, 1077
to_string, 1079, 1080
dds::sub::qos::SubscriberQos, 2423
operator<<, 2427
policy, 2425
to_string, 2425–2427
dds::sub::Query, 2053
add_parameter, 2058
begin, 2056, 2057
data_reader, 2059
end, 2056, 2057
expression, 2056
name, 2058
parameters, 2057, 2058
parameters_length, 2058
Query, 2055, 2056
dds::sub::Rank, 2130
absolute_generation, 2132
generation, 2132
Rank, 2131
sample, 2132
dds::sub::ReadModeDummyType, 2144
dds::sub::Sample< T >, 2270

data, 2272
info, 2273
operator=, 2272, 2273
Sample, 2271, 2272
dds::sub::SampleInfo, 2285
coherent_set_info, 2295
encapsulation_id, 2295
flag, 2293
generation_count, 2289
instance_handle, 2290
original_publication_virtual_guid, 2291
original_publication_virtual_sample_identity, 2292
original_publication_virtual_sequence_number, 2291
publication_handle, 2290
publication_sequence_number, 2291
rank, 2289
reception_sequence_number, 2291
reception_timestamp, 2290
related_original_publication_virtual_guid, 2292
related_original_publication_virtual_sample_identity, 2293
related_original_publication_virtual_sequence_number, 2292
related_source_guid, 2294
related_subscription_guid, 2294
source_guid, 2293
source_timestamp, 2289
state, 2289
topic_query_guid, 2294
valid, 2289
dds::sub::SharedSamples< T, DELEGATE >, 2362
begin, 2364
end, 2364
length, 2364
operator[], 2364
SharedSamples, 2363
unpack, 2365, 2366
dds::sub::status, 686
operator<<, 687, 688
dds::sub::status::DataState, 1118
any, 1124
any_data, 1125
DataState, 1120, 1121
instance_state, 1123, 1124
new_data, 1124
new_instance, 1125
operator!=, 1122
operator<<, 1122, 1125
operator>>, 1122, 1123
operator==, 1121
sample_state, 1123
swap, 1125
view_state, 1124
dds::sub::status::InstanceState, 1615
alive, 1617
any, 1618
InstanceState, 1617
MaskType, 1616
not_alive_disposed, 1617
not_alive_mask, 1618
not_alive_no_writers, 1617
operator<<, 1618
dds::sub::status::SampleState, 2316
any, 2318
MaskType, 2317
not_read, 2318
operator<<, 2318
read, 2318
SampleState, 2317
dds::sub::status::ViewState, 2614
any, 2616
MaskType, 2615
new_view, 2616
not_new_view, 2616
operator<<, 2616
ViewState, 2615
dds::sub::Subscriber, 2410
builtin_subscriber, 2419
close_contained_entities, 2418
default_datareader_qos, 2417
find_subscriber, 2421
find_subscribers, 2419, 2420
get_listener, 2416
implicit_subscriber, 2421
listener, 2415
notify_datareaders, 2414
participant, 2418
qos, 2417
set_listener, 2415, 2416
Subscriber, 2413, 2414
dds::sub::SubscriberListener, 2422
dds::topic, 688
discover_any_topic, 690, 691
discover_topic_data, 691–693
find, 694
get, 690
ignore, 693, 694
dds::topic::AnyTopic, 825
AnyTopic, 826
close, 827
domain_participant, 826
get, 827, 828
inconsistent_topic_status, 826
name, 827
qos, 826, 827
type_name, 827
dds::topic::BuiltinTopicKey, 905
TBuiltinTopicKey, 906

unknown, 906
value, 906
dds::topic::ContentFilteredTopic< T >, 954
append_to_expression_parameter, 958
ContentFilteredTopic, 955
filter, 957
filter_expression, 956
filter_parameters, 956, 957
find_registered_content_filters, 959
remove_from_expression_parameter, 959
topic, 957
dds::topic::Filter, 1558
add_parameter, 1561
begin, 1560, 1561
end, 1561
expression, 1560
Filter, 1559, 1560
name, 1562
parameters, 1561
parameters_length, 1562
dds::topic::is_topic_type< T >, 1622
dds::topic::NoOpTopicListener< T >, 1867
on_inconsistent_topic, 1867
dds::topic::ParticipantBuiltinTopicData, 1908
available_builtin_endpoints_ext, 1912
dds_builtin_endpoints, 1911
default_unicast_locators, 1912
domain_id, 1914
key, 1910
partial_configuration, 1915
participant_name, 1914
partition, 1912
product_version, 1913
property, 1910
reachability_lease_duration, 1915
rtps_protocol_version, 1910
rtps_vendor_id, 1911
TParticipantBuiltinTopicData, 1910
transport_info, 1914
trust_algorithm_info, 1913
trust_protection_info, 1913
user_data, 1910
vendor_builtin_endpoints, 1915
dds::topic::PublicationBuiltinTopicData, 1977
data_tag, 1985
deadline, 1981
destination_order, 1983
disable_positive_acks, 1988
durability, 1980
durability_service, 1981
get_type_no_copy, 1986
group_data, 1984
key, 1979
latency_budget, 1981
lifespan, 1982
liveliness, 1981
locator_filter, 1988
ownership, 1982
ownership_strength, 1983
participant_key, 1980
partition, 1983
presentation, 1983
product_version, 1988
property, 1987
publication_name, 1989
publisher_key, 1986
related_datareader_key, 1990
reliability, 1982
representation, 1984, 1985
rtps_protocol_version, 1987
rtps_vendor_id, 1988
service, 1990
topic_data, 1984
topic_name, 1980
TPublicationBuiltinTopicData, 1979
trust_algorithm_info, 1989
trust_protection_info, 1989
type, 1985
type_name, 1980
unicast_locators, 1987
user_data, 1982
virtual_guid, 1987
dds::topic::qos, 695
operator<<, 698
to_string, 696, 697
dds::topic::qos::TopicQos, 2509
operator<<, 2512, 2515
operator>>, 2513
policy, 2511, 2512
to_string, 2513–2515
TopicQos, 2511
dds::topic::SubscriptionBuiltinTopicData, 2428
content_filter_property, 2436
data_tag, 2434
deadline, 2431
destination_order, 2432
disable_positive_acks, 2437
durability, 2431
get_type_no_copy, 2435
group_data, 2434
key, 2430
latency_budget, 2431
liveliness, 2431
multicast_locators, 2436
ownership, 2432
participant_key, 2430
partition, 2433
presentation, 2433

product_version, 2437
property, 2435
related_datawriter_key, 2438
reliability, 2432
representation, 2434
rtps_protocol_version, 2436
rtps_vendor_id, 2437
service, 2438
subscriber_key, 2435
subscription_name, 2437
time_based_filter, 2433
topic_data, 2433
topic_name, 2430
trust_algorithm_info, 2438
trust_protection_info, 2437
TSubscriptionBuiltinTopicData, 2430
type, 2434
type_name, 2430
unicast_locators, 2436
user_data, 2432
virtual_guid, 2436
dds::topic::Topic< T >, 2474
discover_any_topic, 2485
discover_topic_data, 2485–2487
find, 2489
find_topics, 2490, 2491
get_listener, 2483
ignore, 2487, 2489
inconsistent_topic_status, 2484
listener, 2482
qos, 2483, 2484
set_listener, 2482, 2483
Topic, 2477–2481
dds::topic::topic_type_name< T >, 2493
dds::topic::topic_type_support< T >, 2493
dds::topic::TopicBuiltinTopicData, 2493
deadline, 2496
destination_order, 2498
durability, 2495
durability_service, 2496
history, 2498
key, 2495
latency_budget, 2496
lifespan, 2498
liveliness, 2497
name, 2495
ownership, 2499
reliability, 2497
representation, 2500
resource_limits, 2499
topic_data, 2499
transport_priority, 2497
type_name, 2495
dds::topic::TopicDescription< T >, 2503
name, 2504
participant, 2504
type_name, 2504
dds::topic::TopicInstance< T >, 2505
handle, 2507
operator dds::core::InstanceHandle, 2506
sample, 2507
TopicInstance, 2506
dds::topic::TopicListener< T >, 2508
on_inconsistent_topic, 2509
dds_builtin_endpoints
dds::topic::ParticipantBuiltinTopicData, 1911
DEADLINE, 523
Deadline
dds::core::policy::Deadline, 1257
deadline
dds::topic::PublicationBuiltinTopicData, 1981
dds::topic::SubscriptionBuiltinTopicData, 2431
dds::topic::TopicBuiltinTopicData, 2496
DEBUG
rti::config::PrintFormat_def, 1961
debug
Logging, 458
rti::config::Logger, 1690, 1692
Debuggability, 172
dedicated_participant
rti::core::MonitoringDistributionSettings, 1719
DEFAULT
rti::config::PrintFormat_def, 1961
rti::core::xtypes::DynamicDataTypeSerializationProperty, 1503
rti::topic::PrintFormatKind_def, 1962
rti::util::heap_monitoring::SnapshotContentFormat_def, 2371
Default
dds::core::QosProvider, 2036
rti::topic::PrintFormatProperty, 1966
default_datareader_qos
dds::domain::DomainParticipant, 1348
dds::sub::Subscriber, 2417
default_datawriter_qos
dds::domain::DomainParticipant, 1346, 1347
dds::pub::Publisher, 1998
default_domain_announcement_period
rti::core::policy::DiscoveryConfig, 1300
default_filter_state
dds::sub::DataReader< T >, 991
DEFAULT_LABEL
dds::core::xtypes::UnionMember, 2583
default_library
dds::core::QosProvider, 2041, 2042
default_mask
rti::config::activity_context::AttributeKindMask, 867
rti::config::compliance::RpcMask, 2221

rti::config::compliance::XTypesMask, 2674
rti::util::network_capture::ContentKindMask, 964
rti::util::network_capture::TrafficKindMask, 2532
WIRE_PROTOCOL, 557
DEFAULT_NAME
Flow Controllers, 58
default_participant_qos
dds::domain::DomainParticipant, 1341
default_profile
dds::core::QosProvider, 2041, 2042
default_profile_library
dds::core::QosProvider, 2042
default_provider_params
dds::core::QosProvider, 2036, 2037
default_publication
rti::core::CompressionIdMask, 942
default_publisher_qos
dds::domain::DomainParticipant, 1342
default_qos_provider_params
dds::core::QosProvider, 2048
rti::core, 710
default_subscriber_qos
dds::domain::DomainParticipant, 1343
default_subscription
rti::core::CompressionIdMask, 942
default_topic_qos
dds::domain::DomainParticipant, 1344, 1345
default_unicast_locators
dds::topic::ParticipantBuiltinTopicData, 1912
delegate
dds::core::Value< D >, 2601, 2602
delete_data
dds::pub::DataWriter< T >, 1183
rti::flat::Sample< OffsetType >, 2277
delete_durable_subscription
dds::domain::DomainParticipant, 1357
delete_journal
DURABILITY, 530
deny_interfaces_list
NDDS_Transport_Property_t, 1790
deny_interfaces_list_length
NDDS_Transport_Property_t, 1791
deny_multicast_interfaces_list
NDDS_Transport_Property_t, 1792
deny_multicast_interfaces_list_length
NDDS_Transport_Property_t, 1792
Deprecated QoS, 296
depth
dds::core::policy::History, 1605
deserialized_type_object_dynamic_allocation_threshold
rti::core::policy::DomainParticipantResourceLimits, 1420, 1421
DESTINATION_ORDER, 524
DestinationOrderKind, 524
DestinationOrderScopeKind, 525
destination_order
dds::topic::PublicationBuiltinTopicData, 1983
dds::topic::SubscriptionBuiltinTopicData, 2432
dds::topic::TopicBuiltinTopicData, 2498
DestinationOrder
dds::core::policy::DestinationOrder, 1260, 1261
DestinationOrderKind
DESTINATION_ORDER, 524
DestinationOrderScopeKind
DESTINATION_ORDER, 525
detach_all
dds::core::cond::WaitSet, 2626
detach_condition
dds::core::cond::WaitSet, 2625
rti::core::cond::AsyncWaitSet, 849, 850
detach_reader
rti::sub::SampleProcessor, 2309
detached_instance_count
rti::core::status::DataReaderCacheStatus, 1052
detached_instance_count_peak
rti::core::status::DataReaderCacheStatus, 1053
dheader_in_non_primitive_collections
rti::config::compliance::XTypesMask, 2669
difference_type
rti::core::bounded_sequence< T, MaxLength >, 891
rti::flat::SequenceIterator< E, OffsetKind >, 2332
dimension
dds::core::xtypes::ArrayType, 832
dimension_count
dds::core::xtypes::ArrayType, 832
direct_communication
dds::core::policy::Durability, 1438, 1439
disable
Heap Monitoring, 592
Network Capture, 581
disable_asynchronous_batch
rti::core::policy::AsynchronousPublisher, 836, 837
disable_asynchronous_write
rti::core::policy::AsynchronousPublisher, 835
disable_fragmentation_support
rti::core::policy::BuiltinTopicReaderResourceLimits, 913
rti::core::policy::DataReaderResourceLimits, 1092
disable_inline_keyhash
rti::core::policy::DataWriterProtocol, 1214, 1215
disable_interface_tracking
NDDS_Transport_UDPv4_Property_t, 1807
NDDS_Transport_UDPv4_WAN_Property_t, 1815
NDDS_Transport_UDPv6_Property_t, 1827
disable_positive_acks
dds::topic::PublicationBuiltinTopicData, 1988
dds::topic::SubscriptionBuiltinTopicData, 2437
rti::core::policy::DataReaderProtocol, 1062, 1063

rti::core::policy::DataWriterProtocol, 1214
disable_positive_acks_decrease_sample_keep_duration_factor
rti::core::policy::RtpsReliableWriterProtocol, 2248
disable_positive_acks_enable_adaptive_sample_keep_duration
rti::core::policy::RtpsReliableWriterProtocol, 2247
disable_positive_acks_increase_sample_keep_duration_factor
rti::core::policy::RtpsReliableWriterProtocol, 2248, 2249
disable_positive_acks_max_sample_keep_duration
rti::core::policy::RtpsReliableWriterProtocol, 2246
disable_positive_acks_min_sample_keep_duration
rti::core::policy::RtpsReliableWriterProtocol, 2245, 2246
disable_repair_piggyback_heartbeat
rti::core::policy::RtpsReliableWriterProtocol, 2255
disable_topic_query_publication
rti::core::policy::AsynchronousPublisher, 838
Disabled
rti::core::policy::AsynchronousPublisher, 840
rti::core::policy::Batch, 881
rti::core::policy::Monitoring, 1707
disabled_metrics_selection
rti::core::MonitoringMetricSelection, 1739, 1740
DISALLOW_TYPE_COERCION
dds::core::policy::TypeConsistencyEnforcementKind_def, 2571
DisallowTypeCoercion
dds::core::policy::TypeConsistencyEnforcement, 2569
discard
rti::flat::AbstractBuilder, 784
discard_builder
FlatData Builders, 411
discard_loan
dds::pub::DataWriter< T >, 1185
discover_any_topic
dds::topic, 690, 691
dds::topic::Topic< T >, 2485
discover_topic_data
dds::topic, 691–693
dds::topic::Topic< T >, 2485–2487
discovered_participant_data
dds::domain, 638
dds::domain::DomainParticipant, 1362
discovered_participant_subject_name
dds::domain::DomainParticipant, 1365
rti::domain, 728
discovered_participants
dds::domain, 637, 638
dds::domain::DomainParticipant, 1361, 1362
discovered_participants_from_subject_name
dds::domain::DomainParticipant, 1366
rti::domain, 730
DISCOVERY, 525
Discovery, 177
rti::core::policy::Discovery, 1267
discovery
rti::config::LogCategory_def, 1683
Discovery Isolation, 183
Discovery Metadata, 288
Discovery Snapshot, 573
take_snapshot, 573–577
DISCOVERY_CONFIG, 526
all, 527
none, 527
RemoteParticipantPurgeKind, 526
discovery_service_sample
rti::core::SampleFlag, 2281
DiscoveryConfig
rti::core::policy::DiscoveryConfig, 1279
DiscoveryConfigBuiltinChannelKindMask
rti::core::policy::DiscoveryConfigBuiltinChannelKindMask, 1320
DiscoveryConfigBuiltinPluginKindMask
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask, 1323
discriminator
dds::core::xtypes::UnionType, 2587
discriminator_value
dds::core::xtypes::DynamicData, 1477
DiscriminatorType
dds::core::xtypes::UnionMember, 2579
dds::core::xtypes::UnionType, 2585
dispatch
dds::core::cond::Condition, 949
dds::core::cond::WaitSet, 2623, 2624
dispose_instance
dds::pub::DataWriter< T >, 1162, 1163, 1173
DISPOSED
rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_c 1251
disposed
dds::sub::GenerationCount, 1590
disposed_instance_count
rti::core::status::DataReaderCacheStatus, 1052
rti::core::status::DataWriterCacheStatus, 1201
disposed_instance_count_peak
rti::core::status::DataReaderCacheStatus, 1052
rti::core::status::DataWriterCacheStatus, 1201
disposed_instance_removal
rti::core::DataReaderResourceLimitsInstanceReplacementSettings, 1110
DISPOSED_THEN_ALIVE
rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_c 1251
distribution_settings
rti::core::policy::Monitoring, 1710, 1711
dns_tracker_polling_period

dds::domain::DomainParticipant, 1355
rti::core::policy::DiscoveryConfig, 1311, 1312
Documentation Roadmap, 349
Domain Module, 44
domain_entity_qos_library_name
rti::domain::DomainParticipantConfigParams, 1375, 1376
domain_entity_qos_profile_name
rti::domain::DomainParticipantConfigParams, 1376
domain_id
dds::domain::DomainParticipant, 1338
dds::topic::ParticipantBuiltinTopicData, 1914
rti::config::activity_context::AttributeKindMask, 866
rti::core::MonitoringDedicatedParticipantSettings, 1713, 1714
rti::domain::DomainParticipantConfigParams, 1373
domain_id_gain
rti::core::RtpsWellKnownPorts, 2262
DOMAIN_ID_USE_XML_CONFIG
rti::domain::DomainParticipantConfigParams, 1377
domain_participant
dds::topic::AnyTopic, 826
DOMAIN_PARTICIPANT_RESOURCE_LIMITS, 528
IgnoredEntityReplacementKind, 528
DomainParticipant
dds::domain::DomainParticipant, 1332, 1333
DomainParticipantConfigParams
rti::domain::DomainParticipantConfigParams, 1373
DomainParticipantQos
dds::domain::qos::DomainParticipantQos, 1386
DomainParticipants, 45
DPSE
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask, 1325
drop_incomplete_coherent_set
dds::core::policy::Presentation, 1949, 1950
dropped_content
rti::util::network_capture::NetworkCaptureParams, 1830, 1831
dropped_fragment_count
rti::core::status::DataReaderProtocolStatus, 1071
duplicate_sample_bytes
rti::core::status::DataReaderProtocolStatus, 1068
duplicate_sample_count
rti::core::status::DataReaderProtocolStatus, 1067
DURABILITY, 529
auto_writer_depth, 531
delete_journal, 530
DurabilityKind, 530
full, 531
memory_journal, 530
normal, 531
off, 530, 531
persist_journal, 530
PersistentJournalKind, 530
PersistentSynchronizationKind, 530
truncate_journal, 530
wal_journal, 530
Durability, 184
dds::core::policy::Durability, 1436, 1437
durability
dds::topic::PublicationBuiltinTopicData, 1980
dds::topic::SubscriptionBuiltinTopicData, 2431
dds::topic::TopicBuiltinTopicData, 2495
Durability and Persistence, 99
DURABILITY_SERVICE, 531
durability_service
dds::topic::PublicationBuiltinTopicData, 1981
dds::topic::TopicBuiltinTopicData, 2496
DurabilityKind
DURABILITY, 530
DurabilityService
dds::core::policy::DurabilityService, 1444
Duration
dds::core::Duration, 1450
duration
dds::core::policy::LatencyBudget, 1633
dds::core::policy::Lifespan, 1637
dynamically_allocate_fragmented_samples
rti::core::policy::BuiltinTopicReaderResourceLimits, 916, 917
rti::core::policy::DataReaderResourceLimits, 1095
DynamicData
dds::core::xtypes::DynamicData, 1467, 1468
DynamicDataTypeSerializationProperty
rti::core::xtypes::DynamicDataTypeSerializationProperty, 1501
DynamicType and DynamicData, 439
TypeKind, 440
DynamicType and DynamicData Use Cases, 609
DynamicTypePrintFormatProperty
rti::core::xtypes::DynamicTypePrintFormatProperty, 1510
DynamicTypePrintKind
rti::core::xtypes, 720

EARLIEST_DEADLINE_FIRST
rti::pub::FlowControllerSchedulingPolicy_def, 1582
EarliestDeadlineFirst
rti::pub::FlowControllerProperty, 1580
element_count
rti::core::xtypes::DynamicDataMemberInfo, 1496
rti::flat::AbstractListBuilder< DHeaderGen >, 791
rti::flat::AbstractSequenceBuilder< DHeaderGen >, 795
rti::flat::PrimitiveArrayOffset< T, N >, 1952
rti::flat::PrimitiveSequenceOffset< T >, 1958
rti::flat::SequenceOffset< ElementOffset >, 2344

rti::flat::StringOffset, 2395
element_kind
rti::core::xtypes::DynamicDataMemberInfo, 1497
emergency
Logging, 458
rti::config::Logger, 1689, 1690
empty
rti::core::bounded_sequence< T, MaxLength >, 901
EMPTY_INSTANCES
rti::core::policy::DataReaderInstanceRemovalKind_def, 1054
empty_reliable_writer_cache
rti::core::status::ReliableWriterCacheChangedStatus, 2160
enable
dds::core::Entity, 1524
Heap Monitoring, 591
Network Capture, 581
rti::core::MonitoringDedicatedParticipantSettings, 1713
rti::core::PersistentStorageSettings, 1929
rti::core::policy::Batch, 882
rti::core::policy::Monitoring, 1708
rti::core::policy::TopicQueryDispatch, 2524
rti::queuing::QueueConsumer< T >, 2069
rti::queuing::QueueProducer< T >, 2085
rti::queuing::QueueReplier< TReq, TRep >, 2098
rti::queuing::QueueRequester< TReq, TRep >, 2116
enable_automatic_tss_cleanup
Other Utilities, 596
enable_availability
rti::queuing::QueueConsumerParams, 2078
rti::queuing::QueueReplierParams, 2110
rti::queuing::QueueRequesterParams, 2129
enable_data_consistency_check
rti::core::DataWriterShmemRefTransferModeSettings, 1252, 1253
enable_endpoint_discovery
rti::core::policy::Discovery, 1271
enable_multicast_periodic_heartbeat
rti::core::policy::RtpsReliableWriterProtocol, 2254
enable_required_subscriptions
rti::core::policy::Availability, 871
enable_sample_replication
rti::queuing::QueueProducerParams, 2093
rti::queuing::QueueReplierParams, 2111
rti::queuing::QueueRequesterParams, 2130
enable_udp_debugging
NDDS_Transport_Shmem_Property_t, 1796
enable_v4mapped
NDDS_Transport_UDPv6_Property_t, 1824
enable_wait_for_ack
rti::queuing::QueueProducerParams, 2093
rti::queuing::QueueReplierParams, 2111
rti::queuing::QueueRequesterParams, 2130
Enabled
rti::core::policy::AsynchronousPublisher, 839
rti::core::policy::Batch, 881
rti::core::policy::Monitoring, 1707
enabled_builtin_channels
rti::core::policy::DiscoveryConfig, 1294
enabled_metrics_selection
rti::core::MonitoringMetricSelection, 1738, 1739
enabled_statuses
dds::core::cond::StatusCondition, 2373, 2374
enabled_transports
rti::core::policy::Discovery, 1267
rti::core::policy::TransportSelection, 2555, 2556
EnabledWithMaxDataBytes
rti::core::policy::Batch, 882
EnabledWithMaxSamples
rti::core::policy::Batch, 882
encapsulation_id
dds::sub::SampleInfo, 2295
encapsulation_options_with_padding
rti::config::compliance::XTypesMask, 2670
encrypted
rti::util::network_capture::ContentKindMask, 964
end
dds::core::policy::GroupData, 1593
dds::core::policy::TopicData, 2503
dds::core::policy::UserData, 2594
dds::core::vector< T >, 2610
dds::pub::CoherentSet, 933
dds::sub, 679
dds::sub::CoherentAccess, 931
dds::sub::LoanedSamples< T >, 1668, 1670
dds::sub::Query, 2056, 2057
dds::sub::SharedSamples< T, DELEGATE >, 2364
dds::topic::Filter, 1561
rti::core::bounded_sequence< T, MaxLength >, 899, 900
rti::flat::AbstractAlignedList< ElementOffset >, 782
rti::sub, 766
rti::sub::AckResponseData, 800
rti::sub::ValidLoanedSamples< T >, 2598, 2599
endpoint_type_object_lb_serialization_threshold
rti::core::policy::DiscoveryConfig, 1310, 1311
EndpointGroup
rti::core::EndpointGroup, 1515
EndpointGroupSeq
AVAILABILITY, 516
EndpointTrustAlgorithmInfo
rti::topic::trust::EndpointTrustAlgorithmInfo, 1517
EndpointTrustInterceptorAlgorithmInfo
rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo, 1518
EndpointTrustProtectionInfo

rti::topic::trust::EndpointTrustProtectionInfo, 1519
entities
rti::config::LogCategory_def, 1683
entity
dds::core::cond::StatusCondition, 2374
rti::core::ListenerBinder< Entity, Listener >, 1643
Entity Use Cases, 325
ENTITY_FACTORY, 532
entity_kind
rti::config::activity_context::AttributeKindMask, 866
ENTITY_NAME, 532
entity_name
rti::config::activity_context::AttributeKindMask, 866
rti::queuing::QueueEntityParams< ActualEntity >, 2080
ENTITY_NAME_USE_XML_CONFIG
rti::domain::DomainParticipantConfigParams, 1377
EntityName
rti::core::policy::EntityName, 1531
Entry
dds::core::policy::DataTag, 1134
rti::core::policy::Property, 1972
enum_as_int
rti::topic::PrintFormatProperty, 1964, 1965
enum_as_primitive_in_collections
rti::config::compliance::XTypesMask, 2669
ENUMERATION_TYPE
dds::core::xtypes::TypeKind_def, 2573
EnumMember
dds::core::xtypes::EnumMember, 1533
EnumType
dds::core::xtypes::EnumType, 1536, 1537
erase
rti::core::bounded_sequence< T, MaxLength >, 903
Error
dds::core::Error, 1540
error
Logging, 458
rti::config::Logger, 1689, 1691
evaluate
rti::topic::ContentFilter< T, CompileData >, 953
EVENT, 533
Event
rti::core::policy::Event, 1542
event_settings
rti::core::MonitoringDistributionSettings, 1720
exact_reliability_kind_match
dds::core::policy::Reliability, 2155, 2156
EXCEPTION
rti::config::LogLevel_def, 1693
exception
rti::config::Verbosity_def, 2614
Exceptions, 427, 601
EXCLUSIVE
dds::core::policy::OwnershipKind_def, 1906
Exclusive
dds::core::policy::Ownership, 1905
exists
dds::core::policy::DataTag, 1135
rti::core::policy::Property, 1973
expects_inline_qos
rti::core::policy::DataReaderProtocol, 1062
expiration_code
rti::core::status::InvalidLocalAdvanceNoticeStatus, 1621
expiration_message
rti::core::status::InvalidLocalAdvanceNoticeStatus, 1621
expiration_time
rti::core::status::InvalidLocalAdvanceNoticeStatus, 1621
expired_dropped_sample_count
rti::core::status::DataReaderCacheStatus, 1049
expression
dds::sub::cond::QueryCondition, 2062
dds::sub::Query, 2056
dds::topic::Filter, 1560
expression_parameters
rti::core::ContentFilterProperty, 961
ExpressionProperty
rti::topic::ExpressionProperty, 1548
extensibility_kind
dds::core::xtypes::AbstractConstructedType< MemberType >, 787
dds::core::xtypes::StructType, 2408
ExtensibilityKind
dds::core::xtypes, 633
EXTENSIBLE
dds::core::xtypes::ExtensibilityKind_def, 1551
extensions
dds::core::Value< D >, 2602
external
dds::core::external< T >, 1553, 1554

facility
rti::config::LogMessage, 1695
fast_heartbeat_period
rti::core::policy::RtpsReliableWriterProtocol, 2235, 2236
FATAL_ERROR
rti::config::LogLevel_def, 1693
file_name
rti::core::PersistentStorageSettings, 1929–1931
fill_array
rti::core, 707
Filter
dds::topic::Filter, 1559, 1560
rti::core::policy::LocatorFilter, 1677

filter
dds::topic::ContentFilteredTopic< T >, 957
rti::sub::TopicQuerySelection, 2528
Filter Use Cases, 331
filter_class_name
rti::core::ContentFilterProperty, 961
filter_expression
dds::topic::ContentFilteredTopic< T >, 956
rti::core::ChannelSettings, 923
rti::core::ContentFilterProperty, 961
rti::core::LocatorFilterElement, 1680
filter_name
rti::core::policy::LocatorFilter, 1678
rti::core::policy::MultiChannel, 1752
filter_parameters
dds::topic::ContentFilteredTopic< T >, 956, 957
filtered_sample_bytes
rti::core::status::DataReaderProtocolStatus, 1068
rti::core::status::DataWriterProtocolStatus, 1221
filtered_sample_count
rti::core::status::DataReaderProtocolStatus, 1068
rti::core::status::DataWriterProtocolStatus, 1221
FilterSeq
rti::core::policy::LocatorFilter, 1677
FINAL
dds::core::xtypes::ExtensibilityKind_def, 1551
finalize
rti::topic::ContentFilter< T, CompileData >, 953
finalize_participant_factory
dds::domain::DomainParticipant, 1341
find
dds::domain, 639
dds::domain::DomainParticipant, 1363
dds::pub, 651, 652
dds::pub::DataWriter< T >, 1192, 1193
dds::sub, 672, 673, 675, 677
dds::sub::DataReader< T >, 1037–1040
dds::topic, 694
dds::topic::Topic< T >, 2489
find_content_filter
Custom Content Filters, 572
find_datareader_by_name
dds::sub::DataReader< T >, 1043, 1045
rti::sub, 762, 763
find_datareader_by_topic_description
dds::sub::DataReader< T >, 1044
rti::sub, 763
find_datareader_by_topic_name
dds::sub::DataReader< T >, 1042
rti::sub, 760
find_datareaders
dds::sub::DataReader< T >, 1040, 1041
rti::sub, 759, 760
find_datawriter_by_name
dds::pub::DataWriter< T >, 1197, 1198
rti::pub, 748, 749
find_datawriter_by_topic_name
dds::pub::DataWriter< T >, 1196
rti::pub, 747
find_datawriters
dds::pub::DataWriter< T >, 1194, 1195
rti::pub, 746
find_flow_controller
rti::pub, 750
rti::pub::FlowController, 1576
find_member_by_id
dds::core::xtypes::StructType, 2408
dds::core::xtypes::UnionType, 2587
find_member_by_label
dds::core::xtypes::UnionType, 2587
find_member_by_name
dds::core::xtypes::AbstractConstructedType< MemberType >, 788
find_member_by_ordinal
dds::core::xtypes::EnumType, 1538
find_participant_by_name
dds::domain::DomainParticipant, 1367
rti::domain, 731
find_participants
dds::domain::DomainParticipant, 1367, 1368
rti::domain, 731, 732
find_publisher
dds::pub::Publisher, 2004
rti::pub, 744
find_publishers
dds::pub::Publisher, 2003
rti::pub, 743, 744
find_registered_content_filters
dds::topic::ContentFilteredTopic< T >, 959
rti::topic, 775, 776
find_subscriber
dds::sub::Subscriber, 2421
rti::sub, 758
find_subscribers
dds::sub::Subscriber, 2419, 2420
rti::sub, 757, 758
find_topic_query
rti::sub, 769
rti::sub::TopicQuery, 2519
find_topics
dds::topic::Topic< T >, 2490, 2491
rti::topic, 774
find_type
dds::domain::DomainParticipant, 1369
rti::domain, 732
finish
MyFlatMutableBuilder, 1766
MyFlatUnionBuilder, 1779

rti::flat::FinalSequenceBuilder< ElementOffset >, 1570
rti::flat::MutableArrayBuilder< ElementBuilder, N >, 1754
rti::flat::MutableSequenceBuilder< ElementBuilder >, 1758
rti::flat::PrimitiveSequenceBuilder< T >, 1957
rti::flat::StringBuilder, 2393
finish_sample
MyFlatMutableBuilder, 1766
MyFlatUnionBuilder, 1779
first_available_sample_sequence_number
rti::core::status::DataReaderProtocolStatus, 1070
rti::core::status::DataWriterProtocolStatus, 1223
first_available_sample_virtual_sequence_number
rti::core::status::DataWriterProtocolStatus, 1224
first_unacknowledged_sample_sequence_number
rti::core::status::DataWriterProtocolStatus, 1224
first_unacknowledged_sample_subscription_handle
rti::core::status::DataWriterProtocolStatus, 1225
first_unacknowledged_sample_virtual_sequence_number
rti::core::status::DataWriterProtocolStatus, 1224
first_unelapsed_keep_duration_sample_sequence_number
rti::core::status::DataWriterProtocolStatus, 1225
FIXED_RATE_NAME
Flow Controllers, 59
flag
dds::sub::SampleInfo, 2293
rti::pub::WriteParams, 2649
FlatData Builders, 409
build_data, 411
discard_builder, 411
FlatData Offsets, 414
plain_cast, 417, 418
FlatData Samples, 412
MyFlatFinal, 413
MyFlatMutable, 413
MyFlatUnion, 413
FlatData Topic-Types, 419
FLOAT_128_TYPE
dds::core::xtypes::TypeKind_def, 2573
FLOAT_32_TYPE
dds::core::xtypes::TypeKind_def, 2573
FLOAT_64_TYPE
dds::core::xtypes::TypeKind_def, 2573
floating_point
rti::core::ThreadSettingsKindMask, 2456
Flow Control, 188
Flow Controllers, 57
DEFAULT_NAME, 58
FIXED_RATE_NAME, 59
FlowControllerSchedulingPolicy, 58
ON_DEMAND_NAME, 60
flow_controller_allocation
rti::core::policy::DomainParticipantResourceLimits, 1406
flow_controller_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1413
flow_controller_name
rti::core::policy::PublishMode, 2016
FlowController
rti::pub::FlowController, 1573
FlowControllerProperty
rti::pub::FlowControllerProperty, 1578
FlowControllerSchedulingPolicy
Flow Controllers, 58
FlowControllerTokenBucketProperty
rti::pub::FlowControllerTokenBucketProperty, 1584
flush
dds::pub::DataWriter< T >, 1178
Foo, 1588
force_interface_poll_detection
NDDS_Transport_UDPv4_Property_t, 1806
NDDS_Transport_UDPv4_WAN_Property_t, 1815
NDDS_Transport_UDPv6_Property_t, 1826
force_type_validation
dds::core::policy::TypeConsistencyEnforcement, 2568
Fragmentation, 191
frame_queue_size
rti::util::network_capture::NetworkCaptureParams, 1834
from_cdr_buffer
dds::core::xtypes::DynamicData, 1490
rti::core::xtypes, 721
Topic-type serialization and deserialization, 569
from_cdr_buffer_no_alloc
Topic-type serialization and deserialization, 568
from_microsecs
dds::core::Duration, 1451
dds::core::Time, 2459
from_millisecs
dds::core::Duration, 1451
dds::core::Time, 2460
from_secs
dds::core::Duration, 1452
dds::core::Time, 2460
from_string
dds::core::xtypes::DynamicData, 1489
Topic traits and data-type support, 447
front
rti::core::bounded_sequence< T, MaxLength >, 898
full
DURABILITY, 531
full_reliable_writer_cache
rti::core::status::ReliableWriterCacheChangedStatus, 2160

FULLY_PROCESSED_INSTANCES
rti::core::policy::DataReaderInstanceRemovalKind_def, 1055
FUNCTION
rti::util::heap_monitoring::SnapshotContentFormat_def, 2371
function_call_max_wait
dds::rpc::ClientParams, 929
Functional Categories, 156

gather_send_buffer_count_max
NDDS_Transport_Property_t, 1789
General Utilities and Compliance Configuration, 90
generation
dds::sub::Rank, 2132
generation_count
dds::sub::SampleInfo, 2289
GenerationCount
dds::sub::GenerationCount, 1589
generic_510_transport_compatibility
Builtin Qos Profiles, 472
generic_auto_tuning
Builtin Qos Profiles, 478, 488
generic_best_effort
Builtin Qos Profiles, 474, 483
generic_common
Builtin Qos Profiles, 471
generic_connext_micro_compatibility
Builtin Qos Profiles, 471
generic_connext_micro_compatibility_2_4_3
Builtin Qos Profiles, 472
generic_connext_micro_compatibility_2_4_9
Builtin Qos Profiles, 472
generic_keep_last_reliable
Builtin Qos Profiles, 473, 483
generic_keep_last_reliable_large_data
Builtin Qos Profiles, 476, 485
generic_keep_last_reliable_large_data_fast_flow
Builtin Qos Profiles, 477, 486
generic_keep_last_reliable_large_data_medium_flow
Builtin Qos Profiles, 477, 487
generic_keep_last_reliable_large_data_slow_flow
Builtin Qos Profiles, 477, 487
generic_keep_last_reliable_persistent
Builtin Qos Profiles, 478, 488
generic_keep_last_reliable_transient
Builtin Qos Profiles, 478, 487
generic_keep_last_reliable_transient_local
Builtin Qos Profiles, 477, 487
generic_minimal_memory_footprint
Builtin Qos Profiles, 478, 488
generic_monitoring2
Builtin Qos Profiles, 479
generic_monitoring2_wan
Builtin Qos Profiles, 479
generic_monitoring_common
Builtin Qos Profiles, 471
generic_other_dds_vendor_compatibility
Builtin Qos Profiles, 472
generic_participant_large_data
Builtin Qos Profiles, 474, 484
generic_participant_large_data_monitoring
Builtin Qos Profiles, 475, 484
generic_security
Builtin Qos Profiles, 473
generic_strict_reliable
Builtin Qos Profiles, 473, 483
generic_strict_reliable_high_throughput
Builtin Qos Profiles, 474, 483
generic_strict_reliable_large_data
Builtin Qos Profiles, 475, 485
generic_strict_reliable_large_data_fast_flow
Builtin Qos Profiles, 476, 486
generic_strict_reliable_large_data_medium_flow
Builtin Qos Profiles, 476, 486
generic_strict_reliable_large_data_slow_flow
Builtin Qos Profiles, 476, 486
generic_strict_reliable_low_latency
Builtin Qos Profiles, 474, 484
get
dds::core::external< T >, 1555, 1556
dds::core::optional< T >, 1886
dds::core::policy::DataTag, 1135
dds::pub, 647
dds::pub::AnyDataWriter, 819, 820
dds::sub, 661
dds::sub::AnyDataReader, 810, 811
dds::topic, 690
dds::topic::AnyTopic, 827, 828
rti::core::ListenerBinder< Entity, Listener >, 1643
rti::core::optional_value< T >, 1896
rti::core::pointer< T >, 1939
rti::core::policy::Property, 1973
rti::core::Result< T >, 2211, 2212
rti::core::xtypes::LoanedDynamicData, 1658
rti::flat::PrimitiveConstOffset< T >, 1953
rti::topic::CustomFilter< T >, 969
get_all
dds::core::policy::DataTag, 1137
rti::core::policy::Property, 1975
get_bitbound
dds::core::xtypes::Member, 1704
get_buffer
rti::flat::OffsetBase, 1877
get_buffer_size
rti::flat::OffsetBase, 1877
get_cdr_buffer
dds::core::xtypes::DynamicData, 1481

get_discriminator
dds::core::xtypes::DynamicData, 1478
get_element
rti::flat::AbstractPrimitiveList< T >, 792
rti::flat::FinalAlignedArrayOffset< ElementOffset, N >, 1566
rti::flat::FinalArrayOffset< ElementOffset, N >, 1568
rti::flat::MutableArrayOffset< ElementOffset, N >, 1756
rti::flat::SequenceOffset< ElementOffset >, 2344
get_id
dds::core::xtypes::Member, 1703
dds::core::xtypes::UnionMember, 2581
get_if_ok
rti::core::Result< T >, 2212, 2213
get_listener
dds::domain::DomainParticipant, 1336
dds::pub::DataWriter< T >, 1170
dds::pub::Publisher, 2000
dds::sub::DataReader< T >, 1007
dds::sub::Subscriber, 2416
dds::topic::Topic< T >, 2483
rti::queuing::QueueConsumer< T >, 2068
rti::queuing::QueueProducer< T >, 2084
rti::queuing::QueueReplier< TReq, TRep >, 2098
rti::queuing::QueueRequester< TReq, TRep >, 2115
rti::request::Replier< RequestType, ReplyType >, 2177
get_loan
dds::pub::DataWriter< T >, 1183
get_ptr
dds::core::optional< T >, 1887
get_return_code
rti::core::Result< T >, 2212
get_rpc_mask
Compliance Configuration, 453
get_shared_ptr
dds::core::external< T >, 1556
get_speed
rpc_example::RobotControl, 2215
get_speed_async
rpc_example::RobotControlAsync, 2217
get_status
dds::core::status, 631
dds::core::status::StatusMask, 2389
get_string
rti::flat::StringOffset, 2395
get_tuple
dds::core::xtypes::DynamicData, 1493
rti::core::xtypes, 724
get_type_no_copy
dds::topic::PublicationBuiltinTopicData, 1986
dds::topic::SubscriptionBuiltinTopicData, 2435
get_values
dds::core::xtypes::DynamicData, 1471–1473
get_write_params_for_related_request
rti::queuing::QueueReplier< TReq, TRep >, 2104
get_xtypes_mask
Compliance Configuration, 451
GROUP
dds::core::policy::PresentationAccessScopeKind_def, 1951
group_coherent_set_sequence_number
rti::core::CoherentSetInfo, 937, 938
GROUP_DATA, 534
group_data
dds::topic::PublicationBuiltinTopicData, 1984
dds::topic::SubscriptionBuiltinTopicData, 2434
group_guid
rti::core::CoherentSetInfo, 936
GroupAccessScope
dds::core::policy::Presentation, 1949
GroupData
dds::core::policy::GroupData, 1592
Guid
rti::core::Guid, 1597
guid
rti::queuing::QueueConsumer< T >, 2070
rti::queuing::QueueProducer< T >, 2089
rti::queuing::QueueReplier< TReq, TRep >, 2106
rti::queuing::QueueRequester< TReq, TRep >, 2125
rti::sub::TopicQuery, 2518
guid_prefix
rti::config::activity_context::AttributeKindMask, 865

handle
dds::topic::TopicInstance< T >, 2507
rti::pub::WriteParams, 2648
handler
dds::core::cond::GuardCondition, 1594
dds::core::cond::StatusCondition, 2374
has_bitbound
dds::core::xtypes::Member, 1704
has_id
dds::core::xtypes::Member, 1703
dds::core::xtypes::UnionMember, 2581
has_matching_reader_queue
rti::queuing::QueueConsumer< T >, 2075
rti::queuing::QueueProducer< T >, 2089
has_matching_reply_reader_queue
rti::queuing::QueueReplier< TReq, TRep >, 2102
rti::queuing::QueueRequester< TReq, TRep >, 2125
has_matching_request_reader_queue
rti::queuing::QueueReplier< TReq, TRep >, 2102
rti::queuing::QueueRequester< TReq, TRep >, 2124
has_parent
dds::core::xtypes::StructType, 2408
has_value

dds::core::optional< T >, 1884
rti::core::optional_value< T >, 1894
Heap Monitoring, 589
disable, 592
enable, 591
pause, 592
resume, 593
SnapshotContentFormat, 590
SnapshotOutputFormat, 590
take_snapshot, 593
HeapMonitoringParams
rti::util::heap_monitoring::HeapMonitoringParams, 1600
heartbeat_period
rti::core::policy::RtpsReliableWriterProtocol, 2234, 2235
heartbeat_suppression_duration
rti::core::RtpsReliableReaderProtocol, 2224, 2225
heartbeats_per_max_samples
rti::core::policy::RtpsReliableWriterProtocol, 2240, 2241
high
rti::core::SequenceNumber, 2339
high_watermark
rti::core::policy::RtpsReliableWriterProtocol, 2233, 2234
high_watermark_reliable_writer_cache
rti::core::status::ReliableWriterCacheChangedStatus, 2161
HIGHEST_OFFERED
dds::core::policy::PresentationAccessScopeKind_def, 1951
HIGHEST_PRIORITY_FIRST
rti::pub::FlowControllerSchedulingPolicy_def, 1583
HighestPriorityFirst
rti::pub::FlowControllerProperty, 1580
HISTORY, 533
HistoryKind, 534
History
dds::core::policy::History, 1604
history
dds::topic::TopicBuiltinTopicData, 2498
history_depth
dds::core::policy::DurabilityService, 1446
history_kind
dds::core::policy::DurabilityService, 1445
HISTORY_SNAPSHOT
rti::sub::TopicQuerySelectionKind_def, 2529
HistoryKind
HISTORY, 534
host_port
NDDS_Transport_UDP_WAN_CommPortsMappingInfo, 1798
id
dds::core::xtypes::Member, 1705
dds::core::xtypes::UnionMember, 2583
identity
rti::pub::WriteParams, 2646
idl
rti::core::xtypes, 720
Ignore
AsyncWaitSet, 508
ignore
dds::domain, 635, 636
dds::domain::DomainParticipant, 1359, 1360
dds::pub, 647, 648
dds::pub::DataWriter< T >, 1186, 1187
dds::sub, 667, 668
dds::sub::DataReader< T >, 1030, 1031
dds::topic, 693, 694
dds::topic::Topic< T >, 2487, 2489
ignore_default_domain_announcements
rti::core::policy::DiscoveryConfig, 1301
ignore_enum_literal_names
dds::core::policy::TypeConsistencyEnforcement, 2569
ignore_environment_profile
rti::core::QosProviderParams, 2052
ignore_loopback_interface
NDDS_Transport_UDPv4_Property_t, 1802
NDDS_Transport_UDPv4_WAN_Property_t, 1811
NDDS_Transport_UDPv6_Property_t, 1822
ignore_member_names
dds::core::policy::TypeConsistencyEnforcement, 2567
ignore_nonrunning_interfaces
NDDS_Transport_UDPv4_Property_t, 1803
NDDS_Transport_UDPv4_WAN_Property_t, 1812
NDDS_Transport_UDPv6_Property_t, 1823
ignore_nonup_interfaces
NDDS_Transport_UDPv4_Property_t, 1802
NDDS_Transport_UDPv4_WAN_Property_t, 1811
ignore_resource_profile
rti::core::QosProviderParams, 2052, 2053
ignore_sequence_bounds
dds::core::policy::TypeConsistencyEnforcement, 2566
ignore_string_bounds
dds::core::policy::TypeConsistencyEnforcement, 2567
ignore_user_profile
rti::core::QosProviderParams, 2051
ignored_entity_allocation
rti::core::policy::DomainParticipantResourceLimits, 1403
ignored_entity_hash_buckets

rti::core::policy::DomainParticipantResourceLimits, 1411
ignored_entity_replacement_kind
rti::core::policy::DomainParticipantResourceLimits, 1429
IgnoredEntityReplacementKind
DOMAIN_PARTICIPANT_RESOURCE_LIMITS, 528
Immutable QoS, 243
implicit_publisher
dds::pub::Publisher, 2005
rti::pub, 738
implicit_subscriber
dds::sub::Subscriber, 2421
rti::sub, 754
in
rti::util::network_capture::TrafficKindMask, 2532
inactivate_nonprogressing_readers
rti::core::policy::RtpsReliableWriterProtocol, 2239, 2240
inactive_count
rti::core::status::ReliableReaderActivityChangedStatus, 2158
include_root_elements
rti::topic::PrintFormatProperty, 1965
incomplete_coherent_set
rti::core::CoherentSetInfo, 938
incomplete_snapshot_topic_query
rti::core::SampleFlag, 2281
inconsistent_topic
dds::core::status::StatusMask, 2379
inconsistent_topic_status
dds::topic::AnyTopic, 826
dds::topic::Topic< T >, 2484
incremental_count
rti::core::AllocationSettings, 805
indent
rti::core::QosPrintFormat, 2022
rti::core::xtypes::DynamicTypePrintFormatProperty, 1510
infinite
dds::core::Duration, 1451
info
dds::core::xtypes::DynamicData, 1484
dds::sub::Sample< T >, 2273
rti::sub::LoanedSample< T >, 1661
informational
Logging, 458
rti::config::Logger, 1690, 1692
InfoType
rti::sub::LoanedSample< T >, 1661
Infrastructure Module, 72
initial_active_topic_queries
rti::core::policy::DataWriterResourceLimits, 1246
initial_batches
rti::core::policy::DataWriterResourceLimits, 1239, 1240
initial_concurrent_blocking_threads
rti::core::policy::DataWriterResourceLimits, 1238
initial_count
rti::core::AllocationSettings, 804
rti::core::policy::Event, 1543
initial_fragmented_samples
rti::core::policy::BuiltinTopicReaderResourceLimits, 914, 915
rti::core::policy::DataReaderResourceLimits, 1093
initial_infos
rti::core::policy::BuiltinTopicReaderResourceLimits, 910
rti::core::policy::DataReaderResourceLimits, 1089, 1090
initial_instances
dds::core::policy::ResourceLimits, 2208
initial_objects_per_thread
rti::core::policy::SystemResourceLimits, 2445, 2446
initial_outstanding_reads
rti::core::policy::BuiltinTopicReaderResourceLimits, 911
rti::core::policy::DataReaderResourceLimits, 1090
initial_participant_announcements
rti::core::policy::DiscoveryConfig, 1282
initial_peers
rti::core::policy::Discovery, 1268
initial_records
rti::core::policy::Database, 974
initial_remote_readers
rti::core::policy::DataWriterResourceLimits, 1244, 1245
initial_remote_virtual_writers
rti::core::policy::DataReaderResourceLimits, 1097, 1098
initial_remote_virtual_writers_per_instance
rti::core::policy::DataReaderResourceLimits, 1099
initial_remote_writers
rti::core::policy::DataReaderResourceLimits, 1088
initial_remote_writers_per_instance
rti::core::policy::DataReaderResourceLimits, 1089
initial_samples
dds::core::policy::ResourceLimits, 2207
rti::core::policy::BuiltinTopicReaderResourceLimits, 909
initial_topic_queries
rti::core::policy::DataReaderResourceLimits, 1102
initial_virtual_sequence_number
rti::core::policy::DataWriterProtocol, 1218
initial_virtual_writers
rti::core::policy::DataWriterResourceLimits, 1243
initial_weak_references
rti::core::policy::Database, 975, 976

initialize_discriminator_to_default
rti::config::compliance::XTypesMask, 2672
initialize_native_array
rti::core, 707
initialize_writer_loaned_sample
rti::core::policy::DataWriterResourceLimits, 1248, 1249
inner_enum
dds::core::safe_enum< def, inner >, 2267
insert
rti::core::bounded_sequence< T, MaxLength >, 903
Installing Transport Plugins, 80
INSTANCE
dds::core::policy::PresentationAccessScopeKind_def, 1951
rti::core::policy::DestinationOrderScopeKind_def, 1265
instance
dds::sub, 665
dds::sub::DataReader< T >, 1028
dds::sub::DataReader< T >::Selector, 2322
rti::config::Logger, 1685
Instance Lifecycle, 193
instance_handle
dds::core::Entity, 1525
dds::sub::SampleInfo, 2290
instance_hash_buckets
dds::core::policy::ResourceLimits, 2208, 2209
instance_id
rti::topic::ServiceRequest, 2359
instance_replacement
rti::core::policy::DataReaderResourceLimits, 1106, 1107
rti::core::policy::DataWriterResourceLimits, 1241
INSTANCE_STATE
rti::core::ServiceRequestId_def, 2362
instance_state
dds::sub::status::DataState, 1123, 1124
rti::sub::status::DataStateEx, 1131
instance_state_consistency_kind
dds::core::policy::Reliability, 2155
InstanceAccessScope
dds::core::policy::Presentation, 1949
InstanceHandle
dds::core::InstanceHandle, 1613
InstanceHandleSeq
dds::core, 619
InstanceState
dds::sub::status::InstanceState, 1617
InstanceStateConsistencyKind
RELIABILITY, 545
INT_16_TYPE
dds::core::xtypes::TypeKind_def, 2572
INT_32_TYPE
dds::core::xtypes::TypeKind_def, 2573
INT_64_TYPE
dds::core::xtypes::TypeKind_def, 2573
INT_8_TYPE
dds::core::xtypes::TypeKind_def, 2573
interceptor
rti::topic::trust::EndpointTrustAlgorithmInfo, 1517
rti::topic::trust::ParticipantTrustAlgorithmInfo, 1917
Interface, 363
NDDS_TRANSPORT_INTERFACE_OFF, 365
NDDS_TRANSPORT_INTERFACE_ON, 365
NDDS_Transport_Interface_Status_t, 364
interface_poll_period
NDDS_Transport_UDPv4_Property_t, 1806
NDDS_Transport_UDPv4_WAN_Property_t, 1815
NDDS_Transport_UDPv6_Property_t, 1826
InterfaceType
dds::rpc::ServiceEndpoint< Dispatcher >, 2355
intermediate_reply_sequence
rti::core::SampleFlag, 2280
intermediate_topic_query_sample
rti::core::SampleFlag, 2281
Interoperable
WIRE_PROTOCOL, 557
INVALID
rti::core::LocatorKind_def, 1681
rti::core::TransportClassId_def, 2541
Invalid
rti::core::Locator, 1675
invalid
dds::core::Time, 2459
INVALID_ID
dds::core::xtypes::Member, 1706
dds::core::xtypes::UnionMember, 2583
INVALID_INDEX
dds::core::xtypes::AbstractConstructedType< MemberType >, 790
invalid_local_advance_notice
dds::core::status::StatusMask, 2388
is_aggregation_type
dds::core::xtypes, 634
dds::core::xtypes::DynamicType, 1507
is_assignable_from
dds::core::xtypes::DynamicType, 1506
is_bitset
dds::core::xtypes::Member, 1704
is_cdr
dds::core::xtypes::DynamicData, 1481
is_collection_type
dds::core::xtypes, 633
dds::core::xtypes::DynamicType, 1507
is_constructed_type
dds::core::xtypes, 633
dds::core::xtypes::DynamicType, 1507

is_cpp_compatible
rti::flat::OffsetBase, 1876
is_data_consistent
dds::sub::DataReader< T >, 1022
is_key
dds::core::xtypes::Member, 1703
is_license_message
rti::config::LogMessage, 1695
is_locked
dds::core::external< T >, 1557
is_matched_publication_alive
dds::sub::DataReader< T >, 1035
rti::sub, 755
is_matched_subscription_active
dds::pub::DataWriter< T >, 1191
rti::pub, 741
is_member_key
dds::core::xtypes::DynamicData, 1488
is_must_understand
dds::core::xtypes::Member, 1704
is_nested
rti::flat::AbstractBuilder, 784
is_nil
dds::core::InstanceHandle, 1614
is_null
rti::flat::OffsetBase, 1876
rti::flat::SequenceIterator< E, OffsetKind >, 2332
is_ok
rti::core::Result< T >, 2212
is_optional
dds::core::xtypes::Member, 1703
is_pointer
dds::core::xtypes::AliasType, 802
dds::core::xtypes::Member, 1703
dds::core::xtypes::UnionMember, 2582
is_primitive_type
dds::core::xtypes, 633
dds::core::xtypes::DynamicType, 1507
is_sample_app_acknowledged
dds::pub::DataWriter< T >, 1173
is_security_message
rti::config::LogMessage, 1694
is_set
dds::core::optional< T >, 1884
rti::core::optional_value< T >, 1894
is_standalone
rti::core::QosPrintFormat, 2024
is_type_registered
dds::domain::DomainParticipant, 1351
is_valid
rti::flat::AbstractBuilder, 784
IsReplyRelatedPredicate
rti::request::IsReplyRelatedPredicate< T >, 1623
iterator
dds::sub::cond::QueryCondition, 2060
dds::sub::LoanedSamples< T >, 1665
rti::core::bounded_sequence< T, MaxLength >, 892
rti::flat::AbstractAlignedList< ElementOffset >, 782
rti::sub::SharedSamples< T >, 2367
rti::sub::ValidLoanedSamples< T >, 2596
iterator_category
rti::flat::SequenceIterator< E, OffsetKind >, 2331

java_keyhash_750
rti::config::compliance::XTypesMask, 2672
join_multicast_group_timeout
NDDS_Transport_UDPv4_Property_t, 1807
NDDS_Transport_UDPv6_Property_t, 1827
journal_kind
rti::core::PersistentStorageSettings, 1932
JSON
rti::topic::PrintFormatKind_def, 1962
Json
rti::topic::PrintFormatProperty, 1966
json_schema
rti::core::xtypes, 720

KEEP_ALL
dds::core::policy::HistoryKind_def, 1607
KEEP_LAST
dds::core::policy::HistoryKind_def, 1607
keep_minimum_state_for_instances
rti::core::policy::DataReaderResourceLimits, 1101, 1102
KeepAll
dds::core::policy::History, 1605
KeepLast
dds::core::policy::History, 1606
key
dds::core::KeyedBytesTopicType, 1627
dds::core::KeyedStringTopicType, 1630
dds::core::xtypes::Member, 1704
dds::topic::ParticipantBuiltinTopicData, 1910
dds::topic::PublicationBuiltinTopicData, 1979
dds::topic::SubscriptionBuiltinTopicData, 2430
dds::topic::TopicBuiltinTopicData, 2495
key_establishment
rti::topic::trust::ParticipantTrustAlgorithmInfo, 1917
key_only_filter
rti::topic::ExpressionProperty, 1548, 1549
key_value
dds::pub::DataWriter< T >, 1164, 1165
dds::sub::DataReader< T >, 1003, 1004
KeyedBytesTopicType
dds::core::KeyedBytesTopicType, 1626
KeyedStringTopicType
dds::core::KeyedStringTopicType, 1629
Kind
rti::sub::TopicQuerySelection, 2526

kind
dds::core::policy::DestinationOrder, 1261
dds::core::policy::Durability, 1437
dds::core::policy::History, 1604
dds::core::policy::Liveliness, 1649
dds::core::policy::Ownership, 1904, 1905
dds::core::policy::Reliability, 2153
dds::core::policy::TypeConsistencyEnforcement, 2565, 2566
dds::core::xtypes::DynamicType, 1506
rti::core::Locator, 1674
rti::core::policy::PublishMode, 2016
rti::core::policy::Service, 2354
rti::core::policy::TransportMulticast, 2546
rti::sub::TopicQuerySelection, 2528
rti::topic::PrintFormatProperty, 1963

label
dds::core::xtypes::UnionMember, 2582
label_count
dds::core::xtypes::UnionMember, 2582
labels
dds::core::xtypes::UnionMember, 2582
LabelSeq
dds::core::xtypes::UnionMember, 2579
last_available_sample_sequence_number
rti::core::status::DataReaderProtocolStatus, 1070
rti::core::status::DataWriterProtocolStatus, 1224
last_available_sample_virtual_sequence_number
rti::core::status::DataWriterProtocolStatus, 1224
last_committed_sample_sequence_number
rti::core::status::DataReaderProtocolStatus, 1070
last_corrupted_message_timestamp
rti::core::status::DomainParticipantProtocolStatus, 1384
last_instance_handle
dds::core::status::OfferedDeadlineMissedStatus, 1872
dds::core::status::RequestedDeadlineMissedStatus, 2185
dds::core::status::SampleRejectedStatus, 2316
rti::core::status::ReliableReaderActivityChangedStatus, 2159
last_policy_id
dds::core::status::OfferedIncompatibleQosStatus, 1874
dds::core::status::RequestedIncompatibleQosStatus, 2186
last_publication_handle
dds::core::status::LivelinessChangedStatus, 1654
dds::core::status::SubscriptionMatchedStatus, 2440
last_reason
dds::core::status::SampleLostStatus, 2304
dds::core::status::SampleRejectedStatus, 2315
last_request_handle
rti::core::status::ServiceRequestAcceptedStatus, 2361
last_shared_reader_queue
rti::core::SampleFlag, 2280
last_subscription_handle
dds::core::status::PublicationMatchedStatus, 1992
late_joiner_heartbeat_period
rti::core::policy::RtpsReliableWriterProtocol, 2236, 2237
LATENCY_BUDGET, 534
latency_budget
dds::topic::PublicationBuiltinTopicData, 1981
dds::topic::SubscriptionBuiltinTopicData, 2431
dds::topic::TopicBuiltinTopicData, 2496
LatencyBudget
dds::core::policy::LatencyBudget, 1633
lazily_allocate_batch_resources
rti::core::policy::DataReaderResourceLimits, 1105
lease_duration
dds::core::policy::Liveliness, 1649, 1650
LENGTH
rti::core::Guid, 1599
length
dds::core::BytesTopicType, 920
dds::core::KeyedBytesTopicType, 1628
dds::sub::LoanedSamples< T >, 1667
dds::sub::SharedSamples< T, DELEGATE >, 2364
length_auto
Supporting Types and Constants, 438
LENGTH_UNLIMITED
Supporting Types and Constants, 438
level
rti::config::LogMessage, 1694
rti::core::cond::AsyncWaitSetProperty, 860
library_name
Builtin Qos Profiles, 469, 482, 491
LIFESPAN, 535
Lifespan
dds::core::policy::Lifespan, 1636
lifespan
dds::topic::PublicationBuiltinTopicData, 1982
dds::topic::TopicBuiltinTopicData, 2498
Listener, 1637
dds::domain::DomainParticipant, 1331
listener
dds::domain::DomainParticipant, 1334
dds::pub::DataWriter< T >, 1169
dds::pub::Publisher, 1999
dds::sub::DataReader< T >, 1006
dds::sub::Subscriber, 2415
dds::topic::Topic< T >, 2482
rti::core::ListenerBinder< Entity, Listener >, 1643
rti::queuing::QueueConsumer< T >, 2067, 2068

rti::queuing::QueueProducer< T >, 2083, 2084
rti::queuing::QueueReplier< TReq, TRep >, 2097
rti::queuing::QueueRequester< TReq, TRep >, 2115
rti::request::Replier< RequestType, ReplyType >, 2176, 2177
live
rti::sub::status::StreamKind, 2392
LIVELINESS, 535
LivelinessKind, 535
Liveliness, 199
dds::core::policy::Liveliness, 1648
liveliness
dds::topic::PublicationBuiltinTopicData, 1981
dds::topic::SubscriptionBuiltinTopicData, 2431
dds::topic::TopicBuiltinTopicData, 2497
LIVELINESS_BASED
rti::core::policy::RemoteParticipantPurgeKind_def, 2165
liveliness_changed
dds::core::status::StatusMask, 2383
liveliness_changed_status
dds::sub::DataReader< T >, 1011
liveliness_lost
dds::core::status::StatusMask, 2383
liveliness_lost_status
dds::pub::DataWriter< T >, 1171
LivelinessKind
LIVELINESS, 535
load_compliance_masks
Compliance Configuration, 453
load_profiles
dds::core::QosProvider, 2045
loan_value
dds::core::xtypes::DynamicData, 1475–1477
LoanedDynamicData
rti::core::xtypes::LoanedDynamicData, 1658
LoanedSamples
dds::sub::LoanedSamples< T >, 1666
local_publisher_allocation
rti::core::policy::DomainParticipantResourceLimits, 1399
local_publisher_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1407
local_reader_allocation
rti::core::policy::DomainParticipantResourceLimits, 1398
local_reader_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1407
local_subscriber_allocation
rti::core::policy::DomainParticipantResourceLimits, 1399
local_subscriber_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1408
local_topic_allocation
rti::core::policy::DomainParticipantResourceLimits, 1399, 1400
local_topic_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1408
local_writer_allocation
rti::core::policy::DomainParticipantResourceLimits, 1398
local_writer_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1406
Locator
rti::core::Locator, 1674
locator_filter
dds::topic::PublicationBuiltinTopicData, 1988
locator_filters
rti::core::policy::LocatorFilter, 1677, 1678
LOCATOR_REACHABILITY
rti::core::ServiceRequestId_def, 2362
locator_reachability_assert_period
rti::core::policy::DiscoveryConfig, 1305
locator_reachability_change_detection_period
rti::core::policy::DiscoveryConfig, 1306, 1307
locator_reachability_lease_duration
rti::core::policy::DiscoveryConfig, 1306
LOCATORFILTER, 536
LocatorFilter
rti::core::policy::LocatorFilter, 1677
LocatorFilterElement
rti::core::LocatorFilterElement, 1679
LocatorKind
rti::core, 706
locators
rti::core::LocatorFilterElement, 1679, 1680
LocatorSeq
rti::core::Locator, 1675
LogCategory
Logging, 457
LogFacility
Logging, 459
LOGGING, 536
Logging, 201, 455
alert, 458
critical, 458
debug, 458
emergency, 458
error, 458
informational, 458
LogCategory, 457
LogFacility, 459
LogLevel, 456

middleware, 459
notice, 458
PrintFormat, 457
security_event, 459
service, 459
silent, 458
SyslogLevel, 457
SyslogVerbosity, 458
user, 459
Verbosity, 456
warning, 458
Logging and Version, 90
logging_settings
rti::core::MonitoringDistributionSettings, 1721, 1722
LogLevel
Logging, 456
logs
rti::core::MonitoringTelemetryData, 1746
LongDouble
rti::core::LongDouble, 1696
lookup_instance
dds::pub::DataWriter< T >, 1165
dds::sub::DataReader< T >, 1005
lost_by_availability_waiting_time
rti::core::status::SampleLostState, 2301
lost_by_decode_failure
rti::core::status::SampleLostState, 2302
lost_by_deserialization_failure
rti::core::status::SampleLostState, 2302
lost_by_incomplete_coherent_set
rti::core::status::SampleLostState, 2299
lost_by_instances_limit
rti::core::status::SampleLostState, 2299
lost_by_large_coherent_set
rti::core::status::SampleLostState, 2300
lost_by_out_of_memory
rti::core::status::SampleLostState, 2302
lost_by_remote_writers_per_instance_limit
rti::core::status::SampleLostState, 2299
lost_by_remote_writers_per_sample_limit
rti::core::status::SampleLostState, 2301
lost_by_remote_writers_per_virtual_queue_limit
rti::core::status::SampleLostState, 2301
lost_by_samples_limit
rti::core::status::SampleLostState, 2303
lost_by_samples_per_instance_limit
rti::core::status::SampleLostState, 2303
lost_by_samples_per_remote_writer_limit
rti::core::status::SampleLostState, 2300
lost_by_unknown_instance
rti::core::status::SampleLostState, 2302
lost_by_virtual_writers_limit
rti::core::status::SampleLostState, 2300
lost_by_writer
rti::core::status::SampleLostState, 2299
low
rti::core::SequenceNumber, 2339
low_watermark
rti::core::policy::RtpsReliableWriterProtocol, 2232, 2233
low_watermark_reliable_writer_cache
rti::core::status::ReliableWriterCacheChangedStatus, 2160
lz4
rti::core::CompressionIdMask, 943

major_version
rti::config::LibraryVersion, 1634
rti::core::ProductVersion, 1968
rti::core::ProtocolVersion, 1976
MANUAL_BY_PARTICIPANT
dds::core::policy::LivelinessKind_def, 1655
MANUAL_BY_TOPIC
dds::core::policy::LivelinessKind_def, 1655
ManualByParticipant
dds::core::policy::Liveliness, 1650
ManualByTopic
dds::core::policy::Liveliness, 1651
ManuallyDisposeUnregisteredInstances
dds::core::policy::WriterDataLifecycle, 2662
ManuallyEnable
dds::core::policy::EntityFactory, 1529
mapping_function
rti::core::policy::MulticastMapping, 1749
mappings
rti::core::policy::TransportMulticastMapping, 2549
mask
rti::core::policy::TransportBuiltin, 2536
rti::core::ThreadSettings, 2450
MaskType
dds::core::status::StatusMask, 2378
dds::sub::status::InstanceState, 1616
dds::sub::status::SampleState, 2317
dds::sub::status::ViewState, 2615
rti::config::activity_context::AttributeKindMask, 864
rti::config::compliance::RpcMask, 2219
rti::config::compliance::XTypesMask, 2668
rti::core::CompressionIdMask, 941
rti::core::policy::DiscoveryConfigBuiltinChannelKindMask, 1320
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask, 1323
rti::core::policy::RtpsReservedPortKindMask, 2256
rti::core::policy::TransportBuiltinMask, 2538
rti::core::SampleFlag, 2279
rti::core::ThreadSettingsKindMask, 2455
rti::sub::status::StreamKind, 2391
rti::util::network_capture::ContentKindMask, 963

rti::util::network_capture::TrafficKindMask, 2530
matched_publication_data
dds::sub, 670
dds::sub::DataReader< T >, 1033, 1036
rti::sub, 756
matched_publication_datareader_protocol_status
dds::sub::DataReader< T >, 1013
matched_publication_participant_data
dds::sub::DataReader< T >, 1034
rti::sub, 755
matched_publications
dds::sub, 668, 669
dds::sub::DataReader< T >, 1031, 1032
matched_replier_count
rti::request::Requester< RequestType, ReplyType >, 2198
matched_requester_count
rti::request::Replier< RequestType, ReplyType >, 2178
matched_subscription_data
dds::pub, 650
dds::pub::DataWriter< T >, 1188, 1192
rti::pub, 742
matched_subscription_datawriter_protocol_status
dds::pub::DataWriter< T >, 1177
matched_subscription_participant_data
dds::pub::DataWriter< T >, 1190
rti::pub, 739
matched_subscriptions
dds::pub, 648, 649
dds::pub::DataWriter< T >, 1187, 1188
matched_subscriptions_locators
dds::pub::DataWriter< T >, 1191
rti::pub, 741
Matching, 293
matching_reader_writer_pair_allocation
rti::core::policy::DomainParticipantResourceLimits, 1402
matching_reader_writer_pair_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1411
matching_writer_reader_pair_allocation
rti::core::policy::DomainParticipantResourceLimits, 1402
matching_writer_reader_pair_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1410
max_active_topic_queries
rti::core::policy::DataWriterResourceLimits, 1246, 1247
max_app_ack_remote_readers
rti::core::policy::DataWriterResourceLimits, 1245
max_app_ack_response_length
rti::core::policy::DataReaderResourceLimits, 1101
max_batches
rti::core::policy::DataWriterResourceLimits, 1240
max_blocking_time
dds::core::policy::Reliability, 2153
max_bytes_per_nack_response
rti::core::policy::RtpsReliableWriterProtocol, 2244, 2245
max_concurrent_blocking_threads
rti::core::policy::DataWriterResourceLimits, 1238
max_count
rti::core::AllocationSettings, 804, 805
rti::core::policy::Event, 1543, 1544
max_data_availability_waiting_time
rti::core::policy::Availability, 871, 872
max_data_bytes
rti::core::policy::Batch, 882, 883
max_endpoint_availability_waiting_time
rti::core::policy::Availability, 872, 873
max_endpoint_group_cumulative_characters
rti::core::policy::DomainParticipantResourceLimits, 1428
max_endpoint_groups
rti::core::policy::DomainParticipantResourceLimits, 1427
max_event_count
rti::core::cond::WaitSetProperty, 2629
max_event_delay
rti::core::cond::WaitSetProperty, 2629, 2630
max_flush_delay
rti::core::policy::Batch, 884, 885
max_fragmented_samples
rti::core::policy::BuiltinTopicReaderResourceLimits, 913, 914
rti::core::policy::DataReaderResourceLimits, 1092, 1093
max_fragmented_samples_per_remote_writer
rti::core::policy::BuiltinTopicReaderResourceLimits, 915
rti::core::policy::DataReaderResourceLimits, 1094
max_fragments_per_sample
rti::core::policy::BuiltinTopicReaderResourceLimits, 916
rti::core::policy::DataReaderResourceLimits, 1094, 1095
max_gather_destinations
rti::core::policy::DomainParticipantResourceLimits, 1413
max_heartbeat_response_delay
rti::core::RtpsReliableReaderProtocol, 2224
max_heartbeat_retries
rti::core::policy::RtpsReliableWriterProtocol, 2239
max_historical_logs
rti::core::MonitoringLoggingDistributionSettings, 1729, 1730

max_infos
rti::core::policy::BuiltinTopicReaderResourceLimits, 911
rti::core::policy::DataReaderResourceLimits, 1087, 1088
max_initial_participant_announcement_period
rti::core::policy::DiscoveryConfig, 1284, 1285
max_instances
dds::core::policy::DurabilityService, 1447
dds::core::policy::ResourceLimits, 2206
max_interface_count
NDDS_Transport_Property_t, 1793
max_liveliness_loss_detection_period
rti::core::policy::DiscoveryConfig, 1281, 1282
max_nack_response_delay
rti::core::policy::RtpsReliableWriterProtocol, 2242, 2243
max_objects_per_thread
rti::core::policy::SystemResourceLimits, 2445, 2446
max_outstanding_reads
rti::core::policy::BuiltinTopicReaderResourceLimits, 912
rti::core::policy::DataReaderResourceLimits, 1091
max_partition_cumulative_characters
rti::core::policy::DomainParticipantResourceLimits, 1417
max_partitions
rti::core::policy::DomainParticipantResourceLimits, 1417
max_query_condition_filters
rti::core::policy::DataReaderResourceLimits, 1100
max_remote_reader_filters
rti::core::policy::DataWriterResourceLimits, 1239
max_remote_readers
rti::core::policy::DataWriterResourceLimits, 1244, 1245
max_remote_virtual_writers
rti::core::policy::DataReaderResourceLimits, 1097
max_remote_virtual_writers_per_instance
rti::core::policy::DataReaderResourceLimits, 1098
max_remote_writers
rti::core::policy::DataReaderResourceLimits, 1085, 1086
max_remote_writers_per_instance
rti::core::policy::DataReaderResourceLimits, 1086
max_remote_writers_per_sample
rti::core::policy::DataReaderResourceLimits, 1099, 1100
max_samples
dds::core::policy::DurabilityService, 1446
dds::core::policy::ResourceLimits, 2205, 2206
dds::sub, 662
dds::sub::DataReader< T >, 1025
dds::sub::DataReader< T >::Selector, 2325
rti::core::policy::Batch, 883, 884
rti::core::policy::BuiltinTopicReaderResourceLimits, 909, 910
max_samples_per_instance
dds::core::policy::DurabilityService, 1447
dds::core::policy::ResourceLimits, 2207
max_samples_per_read
rti::core::policy::BuiltinTopicReaderResourceLimits, 912
rti::core::policy::DataReaderResourceLimits, 1091
max_samples_per_remote_writer
rti::core::policy::DataReaderResourceLimits, 1087
max_send_window_size
rti::core::policy::RtpsReliableWriterProtocol, 2250, 2251
max_size
rti::core::bounded_sequence< T, MaxLength >, 901
max_size_serialized
rti::core::xtypes::DynamicDataTypeSerializationProperty, 1501
max_skiplist_level
rti::core::policy::Database, 974
max_tokens
rti::pub::FlowControllerTokenBucketProperty, 1585
max_topic_queries
rti::core::policy::DataReaderResourceLimits, 1103
max_total_instances
rti::core::policy::DataReaderResourceLimits, 1096
max_virtual_writers
rti::core::policy::DataWriterResourceLimits, 1244
max_weak_references
rti::core::policy::Database, 974, 975
MAXIMAL
rti::config::PrintFormat_def, 1961
maximum
dds::core::Time, 2459
rti::core::SequenceNumber, 2338
Member
dds::core::xtypes::AbstractConstructedType< MemberType >, 786
dds::core::xtypes::Member, 1702
member
dds::core::xtypes::AbstractConstructedType< MemberType >, 787
member_count
dds::core::xtypes::AbstractConstructedType< MemberType >, 787
dds::core::xtypes::DynamicData, 1483
rti::core::xtypes::DynamicDataInfo, 1494
member_exists
dds::core::xtypes::DynamicData, 1483, 1484
rti::core::xtypes::DynamicDataMemberInfo, 1497
member_exists_in_type
dds::core::xtypes::DynamicData, 1484

member_index
dds::core::xtypes::DynamicData, 1487
rti::core::xtypes::DynamicDataMemberInfo, 1496
member_info
dds::core::xtypes::DynamicData, 1484, 1486
member_kind
rti::core::xtypes::DynamicDataMemberInfo, 1496
member_name
rti::core::xtypes::DynamicDataMemberInfo, 1496
member_type
dds::core::xtypes::DynamicData, 1489
MemberIndex
dds::core::xtypes::AbstractConstructedType< MemberType >, 786
members
dds::core::xtypes::AbstractConstructedType< MemberType >, 788
memory_journal
DURABILITY, 530
message_auth
rti::topic::trust::ParticipantTrustSignatureAlgorithmInfo, 1923
message_id
rti::config::LogMessage, 1695
message_size_max
NDDS_Transport_Property_t, 1789
rti::core::TransportInfo, 2543
metatraffic_transport_priority
rti::core::policy::Discovery, 1269, 1270
metrics
rti::core::MonitoringTelemetryData, 1745, 1746
middleware
Logging, 459
Middleware Memory Management, 204
middleware_forwarding_level
rti::core::MonitoringLoggingForwardingSettings, 1734
min_app_ack_response_keep_duration
rti::core::RtpsReliableReaderProtocol, 2227, 2228
min_heartbeat_response_delay
rti::core::RtpsReliableReaderProtocol, 2223
min_initial_participant_announcement_period
rti::core::policy::DiscoveryConfig, 1283, 1284
min_nack_response_delay
rti::core::policy::RtpsReliableWriterProtocol, 2242
min_send_window_size
rti::core::policy::RtpsReliableWriterProtocol, 2249, 2250
min_size_serialized
rti::core::xtypes::DynamicDataTypeSerializationProperty, 1501, 1502
MINIMAL
rti::config::PrintFormat_def, 1961
rti::util::heap_monitoring::SnapshotContentFormat_def, 2371
minimum_separation
dds::core::policy::TimeBasedFilter, 2472
minor_version
rti::config::LibraryVersion, 1634
rti::core::ProductVersion, 1968
rti::core::ProtocolVersion, 1977
MONITORING, 536
Monitoring
rti::core::policy::Monitoring, 1707
Monitoring Library 2.0, 598
RTI_MONITORING_EVENT_TOPIC_NAME, 600
RTI_Monitoring_initialize, 599
RTI_MONITORING_LOGGING_TOPIC_NAME, 600
RTI_MONITORING_PERIODIC_TOPIC_NAME, 600
MONITORING_LIBRARY_COMMAND
rti::core::ServiceRequestId_def, 2362
MONITORING_LIBRARY_REPLY
rti::core::ServiceRequestId_def, 2362
MonitoringDedicatedParticipantSettings
rti::core::MonitoringDedicatedParticipantSettings, 1712
MonitoringDistributionSettings
rti::core::MonitoringDistributionSettings, 1718
MonitoringEventDistributionSettings
rti::core::MonitoringEventDistributionSettings, 1723
MonitoringLoggingDistributionSettings
rti::core::MonitoringLoggingDistributionSettings, 1728
MonitoringMetricSelection
rti::core::MonitoringMetricSelection, 1737
MonitoringMetricSelectionSeq
rti::core, 706
MonitoringPeriodicDistributionSettings
rti::core::MonitoringPeriodicDistributionSettings, 1741
move
dds::sub, 678
dds::sub::LoanedSamples< T >, 1669
Multi-channel DataWriters, 73
multicast_enabled
NDDS_Transport_UDPv4_Property_t, 1801
NDDS_Transport_UDPv6_Property_t, 1822
multicast_locators
dds::topic::SubscriptionBuiltinTopicData, 2436
multicast_loopback_disabled
NDDS_Transport_UDPv4_Property_t, 1802
NDDS_Transport_UDPv6_Property_t, 1822
multicast_receive_addresses
rti::core::policy::Discovery, 1268, 1269
multicast_resend_threshold
rti::core::policy::RtpsReliableWriterProtocol, 2254
multicast_settings

rti::core::ChannelSettings, 922, 923
multicast_ttl
NDDS_Transport_UDPv4_Property_t, 1801
NDDS_Transport_UDPv6_Property_t, 1822
MULTICHANNEL, 537
ChannelSettingsSeq, 538
MultiChannel
rti::core::policy::MultiChannel, 1751
must_understand
dds::core::xtypes::Member, 1705
must_understand_in_key_and_discriminator
rti::config::compliance::XTypesMask, 2673
MUTABLE
dds::core::xtypes::ExtensibilityKind_def, 1551
Mutable QoS, 280
my_complex
MyFlatFinalOffset, 1761, 1762
my_complex_array
MyFlatFinalOffset, 1761, 1762
my_final
MyFlatMutableOffset, 1773, 1775
MyFlatUnionOffset, 1783, 1784
my_final_array
MyFlatMutableOffset, 1773, 1775
my_final_seq
MyFlatMutableOffset, 1773, 1776
my_mutable
MyFlatMutableOffset, 1774, 1776
MyFlatUnionOffset, 1782, 1783
my_mutable_array
MyFlatMutableOffset, 1774, 1776
my_mutable_seq
MyFlatMutableOffset, 1774, 1776
my_optional_primitive
MyFlatMutableOffset, 1772
my_primitive
MyFlatFinalOffset, 1761, 1762
MyFlatMutableOffset, 1772, 1775
MyFlatUnionOffset, 1782, 1783
my_primitive_array
MyFlatFinalOffset, 1761, 1762
MyFlatMutableOffset, 1773, 1775
my_primitive_seq
MyFlatMutableOffset, 1773, 1775
my_string
MyFlatMutableOffset, 1774, 1776
my_string_seq
MyFlatMutableOffset, 1774, 1777
MyFlatFinal
FlatData Samples, 413
MyFlatFinalOffset, 1759
ConstOffset, 1760
my_complex, 1761, 1762
my_complex_array, 1761, 1762
my_primitive, 1761, 1762
my_primitive_array, 1761, 1762
MyFlatFinalOffset, 1761
MyFlatMutable
FlatData Samples, 413
MyFlatMutableBuilder, 1763
add_my_final, 1768
add_my_final_array, 1768
add_my_optional_primitive, 1767
add_my_primitive, 1767
add_my_primitive_array, 1767
build_my_final_seq, 1768
build_my_mutable, 1769
build_my_mutable_array, 1769
build_my_mutable_seq, 1769
build_my_primitive_seq, 1768
build_my_string, 1769
build_my_string_seq, 1769
finish, 1766
finish_sample, 1766
MyFlatMutableBuilder, 1765
Offset, 1765
MyFlatMutableOffset, 1770
ConstOffset, 1771
my_final, 1773, 1775
my_final_array, 1773, 1775
my_final_seq, 1773, 1776
my_mutable, 1774, 1776
my_mutable_array, 1774, 1776
my_mutable_seq, 1774, 1776
my_optional_primitive, 1772
my_primitive, 1772, 1775
my_primitive_array, 1773, 1775
my_primitive_seq, 1773, 1775
my_string, 1774, 1776
my_string_seq, 1774, 1777
MyFlatMutableOffset, 1772
MyFlatUnion
FlatData Samples, 413
MyFlatUnionBuilder, 1777
add_my_final, 1780
add_my_primitive, 1779
build_my_mutable, 1779
finish, 1779
finish_sample, 1779
MyFlatUnionBuilder, 1778
Offset, 1778
MyFlatUnionOffset, 1780
_d, 1782
ConstOffset, 1781
my_final, 1783, 1784
my_mutable, 1782, 1783
my_primitive, 1782, 1783
MyFlatUnionOffset, 1781

nack_period
rti::core::RtpsReliableReaderProtocol, 2225
nack_suppression_duration
rti::core::policy::RtpsReliableWriterProtocol, 2243, 2244
name
dds::core::policy::Partition, 1926
dds::core::xtypes::DynamicType, 1506
dds::core::xtypes::EnumMember, 1534
dds::core::xtypes::Member, 1702, 1704
dds::core::xtypes::UnionMember, 2581, 2583
dds::sub::Query, 2058
dds::topic::AnyTopic, 827
dds::topic::Filter, 1562
dds::topic::TopicBuiltinTopicData, 2495
dds::topic::TopicDescription< T >, 2504
rti::core::policy::EntityName, 1531, 1532
rti::pub::FlowController, 1574
Namespaces and headers, 355
nanosec
dds::core::Duration, 1453
dds::core::Time, 2461
NDDS_DISCOVERY_PEERS, 559
NDDS_Transport_Address_from_string
Transport Address, 375
NDDS_TRANSPORT_ADDRESS_INVALID
Transport Address, 377
NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER
Transport Address, 373
NDDS_Transport_Address_is_ipv4
Transport Address, 376
NDDS_Transport_Address_is_multicast
Transport Address, 377
NDDS_Transport_Address_print
Transport Address, 376
NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE
Transport Address, 374
NDDS_Transport_Address_t, 1784
network_ordered_value, 1785
NDDS_Transport_Address_to_string
Transport Address, 374
NDDS_Transport_Address_to_string_with_protocol_family_format
Transport Address, 375
NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT
Transport Plugins Configuration, 368
NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC
Transport Plugins Configuration, 368
NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED
Transport Plugins Configuration, 367
NDDS_TRANSPORT_CLASSID_INVALID
Transport Plugins Configuration, 368
NDDS_TRANSPORT_CLASSID_RESERVED_RANGE
Transport Plugins Configuration, 370
NDDS_TRANSPORT_CLASSID_SHMEM
Transport Plugins Configuration, 369
NDDS_TRANSPORT_CLASSID_SHMEM_510
Transport Plugins Configuration, 369
NDDS_Transport_ClassId_t
Transport Plugins Configuration, 371
NDDS_TRANSPORT_CLASSID_TCPV4_LAN
Transport Plugins Configuration, 369
NDDS_TRANSPORT_CLASSID_TCPV4_WAN
Transport Plugins Configuration, 369
NDDS_TRANSPORT_CLASSID_TLSV4_LAN
Transport Plugins Configuration, 370
NDDS_TRANSPORT_CLASSID_TLSV4_WAN
Transport Plugins Configuration, 370
NDDS_TRANSPORT_CLASSID_UDPv4
Transport Plugins Configuration, 368
NDDS_TRANSPORT_CLASSID_UDPv4_WAN
Transport Plugins Configuration, 370
NDDS_TRANSPORT_CLASSID_UDPv6
Transport Plugins Configuration, 369
NDDS_TRANSPORT_CLASSID_UDPv6_510
Transport Plugins Configuration, 369
NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN
Transport Plugins Configuration, 370
NDDS_TRANSPORT_INTERFACE_OFF
Interface, 365
NDDS_TRANSPORT_INTERFACE_ON
Interface, 365
NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN
Transport Plugins Configuration, 367
NDDS_Transport_Interface_Status_t
Interface, 364
NDDS_Transport_Interface_t, 1785
address, 1785
rank, 1786
status, 1786
transport_classid, 1785
NDDS_TRANSPORT_LENGTH_UNLIMITED
Transport Plugins Configuration, 367
NDDS_TRANSPORT_PORT_INVALID
Transport Plugins Configuration, 367
NDDS_Transport_Port_t
Transport Plugins Configuration, 371
NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED
Transport Plugins Configuration, 370
NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_M
Transport Plugins Configuration, 371
NDDS_Transport_Property_t, 1786
gather_send_buffer_count, 1788
allow_interfaces_list, 1789
allow_interfaces_list_length, 1790
allow_multicast_interfaces_list, 1791
allow_multicast_interfaces_list_length, 1792
classid, 1787
deny_interfaces_list, 1790

deny_interfaces_list_length, 1791
deny_multicast_interfaces_list, 1792
deny_multicast_interfaces_list_length, 1792
gather_send_buffer_count_max, 1789
max_interface_count, 1793
message_size_max, 1789
properties_bitmap, 1788
thread_name_prefix, 1793
transport_uuid, 1793
NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT
Shared Memory Transport, 384
NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT
Shared Memory Transport, 384
NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_1424_FIX
Shared Memory Transport, 385
NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT
Shared Memory Transport, 384
NDDS_Transport_Shmem_new
Shared Memory Transport, 385
NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT
Shared Memory Transport, 384
NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT
Shared Memory Transport, 385
NDDS_Transport_Shmem_Property_t, 1794
enable_udp_debugging, 1796
parent, 1795
receive_buffer_size, 1795
received_message_count_max, 1795
udp_debugging_address, 1796
udp_debugging_port, 1797
NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT
Shared Memory Transport, 385
NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT
Shared Memory Transport, 384
NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT
UDP Transport Plugin definitions, 379
NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT
UDP Transport Plugin definitions, 380
NDDS_Transport_UDP_Port
UDP Transport Plugin definitions, 380
NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT
UDP Transport Plugin definitions, 378
NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT
UDP Transport Plugin definitions, 380
NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT
UDP Transport Plugin definitions, 379
NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT
UDP Transport Plugin definitions, 379
NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX
UDP Transport Plugin definitions, 379
NDDS_Transport_UDP_WAN_CommPortsMappingInfo, 1797
host_port, 1798
public_port, 1798
rtps_port, 1797
NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT
UDPv4 Transport, 390
NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS
UDPv4 Transport, 392
NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT
UDPv4 Transport, 393
NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER
UDPv4 Transport, 392
NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_D
UDPv4 Transport, 390
NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT
UDPv4 Transport, 392
NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT
UDPv4 Transport, 392
NDDS_Transport_UDPv4_new
UDPv4 Transport, 393
NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX
UDPv4 Transport, 392
NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT
UDPv4 Transport, 390
NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT
UDPv4 Transport, 393
NDDS_Transport_UDPv4_Property_t, 1798
disable_interface_tracking, 1807
force_interface_poll_detection, 1806
ignore_loopback_interface, 1802
ignore_nonrunning_interfaces, 1803
ignore_nonup_interfaces, 1802
interface_poll_period, 1806
multicast_group_timeout, 1807
multicast_enabled, 1801
multicast_loopback_disabled, 1802
multicast_ttl, 1801
no_zero_copy, 1803
parent, 1800
port_offset, 1808
protocol_overhead_max, 1806
public_address, 1807
recv_socket_buffer_size, 1800
reuse_multicast_receive_resource, 1806
send_blocking, 1804
send_ping, 1805
send_socket_buffer_size, 1800
transport_priority_mapping_high, 1805
transport_priority_mapping_low, 1805
transport_priority_mask, 1804
unicast_enabled, 1801
use_checksum, 1804
NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT
UDPv4 Transport, 391
NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT
UDPv4 Transport, 391
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT

UDPv4 Transport, 391
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX
UDPv4 Transport, 391
NDDS_Transport_UDPv4_string_to_address_cEA
UDPv4 Transport, 393
NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT
UDPv4 Transport, 390
NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT
Real-Time WAN Transport, 398
NDDS_Transport_UDPv4_WAN_new
Real-Time WAN Transport, 398
NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT
Real-Time WAN Transport, 398
NDDS_Transport_UDPv4_WAN_Property_t, 1809
binding_ping_period, 1818
comm_ports_list, 1816
comm_ports_list_length, 1817
disable_interface_tracking, 1815
force_interface_poll_detection, 1815
ignore_loopback_interface, 1811
ignore_nonrunning_interfaces, 1812
ignore_nonup_interfaces, 1811
interface_poll_period, 1815
no_zero_copy, 1812
parent, 1810
port_offset, 1818
protocol_overhead_max, 1815
public_address, 1816
recv_socket_buffer_size, 1810
send_blocking, 1813
send_ping, 1814
send_socket_buffer_size, 1810
transport_priority_mapping_high, 1814
transport_priority_mapping_low, 1814
transport_priority_mask, 1813
use_checksum, 1813
NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT
UDPv6 Transport, 403
NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS
UDPv6 Transport, 405
NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER
UDPv6 Transport, 405
NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT
UDPv6 Transport, 404
NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT
UDPv6 Transport, 405
NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT
UDPv6 Transport, 405
NDDS_Transport_UDPv6_new
UDPv6 Transport, 406
NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX
UDPv6 Transport, 405
NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT
UDPv6 Transport, 403
NDDS_Transport_UDPv6_Property_t, 1819
disable_interface_tracking, 1827
enable_v4mapped, 1824
force_interface_poll_detection, 1826
ignore_loopback_interface, 1822
ignore_nonrunning_interfaces, 1823
interface_poll_period, 1826
multicast_group_timeout, 1827
multicast_enabled, 1822
multicast_loopback_disabled, 1822
multicast_ttl, 1822
no_zero_copy, 1823
parent, 1820
port_offset, 1828
protocol_overhead_max, 1826
public_address, 1827
recv_socket_buffer_size, 1821
reuse_multicast_receive_resource, 1826
send_blocking, 1824
send_ping, 1825
send_socket_buffer_size, 1820
transport_priority_mapping_high, 1825
transport_priority_mapping_low, 1825
transport_priority_mask, 1824
unicast_enabled, 1821
NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT
UDPv6 Transport, 404
NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT
UDPv6 Transport, 404
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT
UDPv6 Transport, 404
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX
UDPv6 Transport, 404
NDDS_Transport_UDPv6_string_to_address_cEA
UDPv6 Transport, 405
NDDS_Transport_UUID, 1829
NDDS_TRANSPORT_UUID_SIZE
Transport Plugins Configuration, 367
NDDS_TRANSPORT_UUID_UNKNOWN
Transport Plugins Configuration, 367
Network Capture, 578
disable, 581
enable, 581
pause, 587
resume, 588
set_default_params, 582
start, 583–585
stop, 586
network_ordered_value
NDDS_Transport_Address_t, 1785
new_data
dds::sub::status::DataState, 1124
rti::sub::status::DataStateEx, 1131
new_instance

dds::sub::status::DataState, 1125
rti::sub::status::DataStateEx, 1132
new_remote_participant_announcements
rti::core::policy::DiscoveryConfig, 1283
new_view
dds::sub::status::ViewState, 2616
next_instance
dds::sub, 665
dds::sub::DataReader< T >, 1028
dds::sub::DataReader< T >::Selector, 2322
nil
dds::core::InstanceHandle, 1614
no_compile_data
Custom Content Filters, 572
NO_INSTANCE
rti::core::policy::DataReaderInstanceRemovalKind_def, 1054
NO_PURGE
rti::core::policy::RemoteParticipantPurgeKind_def, 2165
NO_ROTATION
rti::core::ThreadSettingsCpuRotationKind_def, 2454
NO_SERVICE
rti::core::policy::ServiceKind_def, 2358
no_writers
dds::sub::GenerationCount, 1590
no_writers_instance_count
rti::core::status::DataReaderCacheStatus, 1051
no_writers_instance_count_peak
rti::core::status::DataReaderCacheStatus, 1052
no_writers_instance_removal
rti::core::DataReaderResourceLimitsInstanceReplacementSettings, 1110, 1111
no_zero_copy
NDDS_Transport_UDPv4_Property_t, 1803
NDDS_Transport_UDPv4_WAN_Property_t, 1812
NDDS_Transport_UDPv6_Property_t, 1823
NoAutoPurge
dds::core::policy::ReaderDataLifecycle, 2141
None
rti::core::policy::TransportBuiltin, 2535
none
dds::core::status::StatusMask, 2379
DISCOVERY_CONFIG, 527
RELIABILITY, 545
rti::config::activity_context::AttributeKindMask, 867
rti::core::CompressionIdMask, 942
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask, 1324
rti::core::policy::TransportBuiltinMask, 2539
rti::util::network_capture::ContentKindMask, 964
rti::util::network_capture::TrafficKindMask, 2532
WIRE_PROTOCOL, 556
normal
DURABILITY, 531
NoService
rti::core::policy::Service, 2352
not_alive_count
dds::core::status::LivelinessChangedStatus, 1653
not_alive_count_change
dds::core::status::LivelinessChangedStatus, 1653
not_alive_disposed
dds::sub::status::InstanceState, 1617
not_alive_mask
dds::sub::status::InstanceState, 1618
not_alive_no_writers
dds::sub::status::InstanceState, 1617
not_lost
rti::core::status::SampleLostState, 2299
not_new_view
dds::sub::status::ViewState, 2616
not_read
dds::sub::status::SampleState, 2318
not_rejected
dds::core::status::SampleRejectedState, 2312
NOT_SET
rti::core::policy::CdrPaddingKind_def, 921
notice
Logging, 458
rti::config::Logger, 1690, 1691
notify_datareaders
dds::sub::Subscriber, 2414
null
Supporting Types and Constants, 438
null_type
Supporting Types and Constants, 436

Observability Framework, 91
OBSERVABILITY_COLLECTOR
rti::core::policy::ServiceKind_def, 2358
ObservabilityCollectorService
rti::core::policy::Service, 2353
OCTET_TYPE
dds::core::xtypes::TypeKind_def, 2573
off
DURABILITY, 530, 531
offered_deadline_missed
dds::core::status::StatusMask, 2379
offered_deadline_missed_status
dds::pub::DataWriter< T >, 1171
offered_incompatible_qos
dds::core::status::StatusMask, 2380
offered_incompatible_qos_status
dds::pub::DataWriter< T >, 1171
Offset
MyFlatMutableBuilder, 1765
MyFlatUnionBuilder, 1778

rti::flat::MutableArrayBuilder< ElementBuilder, N >, 1754
rti::flat::MutableSequenceBuilder< ElementBuilder >, 1758
rti::flat::Sample< OffsetType >, 2275
old_source_timestamp_dropped_sample_count
rti::core::status::DataReaderCacheStatus, 1048
on_application_acknowledgment
dds::domain::NoOpDomainParticipantListener, 1849
dds::pub::AnyDataWriterListener, 824
dds::pub::DataWriterListener< T >, 1207
dds::pub::NoOpDataWriterListener< T >, 1843
dds::pub::NoOpPublisherListener, 1855
on_data_available
dds::domain::NoOpDomainParticipantListener, 1851
dds::sub::AnyDataReaderListener, 814
dds::sub::DataReaderListener< T >, 1058
dds::sub::NoOpDataReaderListener< T >, 1840
dds::sub::NoOpSubscriberListener, 1866
on_data_on_readers
dds::domain::NoOpDomainParticipantListener, 1850
dds::sub::NoOpSubscriberListener, 1865
on_data_request
dds::pub::NoOpDataWriterListener< T >, 1844
on_data_return
dds::pub::NoOpDataWriterListener< T >, 1844
ON_DEMAND_NAME
Flow Controllers, 60
on_destination_unreachable
dds::pub::NoOpDataWriterListener< T >, 1844
on_inconsistent_topic
dds::domain::NoOpDomainParticipantListener, 1852
dds::topic::NoOpTopicListener< T >, 1867
dds::topic::TopicListener< T >, 2509
on_instance_replaced
dds::domain::NoOpDomainParticipantListener, 1849
dds::pub::AnyDataWriterListener, 824
dds::pub::DataWriterListener< T >, 1206
dds::pub::NoOpDataWriterListener< T >, 1843
dds::pub::NoOpPublisherListener, 1855
on_invalid_local_status_advance_notice
dds::domain::DomainParticipantListener, 1382
dds::domain::NoOpDomainParticipantListener, 1849
on_liveliness_changed
dds::domain::NoOpDomainParticipantListener, 1851
dds::sub::AnyDataReaderListener, 814
dds::sub::DataReaderListener< T >, 1057
dds::sub::NoOpDataReaderListener< T >, 1839
dds::sub::NoOpSubscriberListener, 1866
on_liveliness_lost
dds::domain::NoOpDomainParticipantListener, 1847
dds::pub::AnyDataWriterListener, 822
dds::pub::DataWriterListener< T >, 1204
dds::pub::NoOpDataWriterListener< T >, 1842
dds::pub::NoOpPublisherListener, 1854
on_offered_deadline_missed
dds::domain::NoOpDomainParticipantListener, 1847
dds::pub::AnyDataWriterListener, 822
dds::pub::DataWriterListener< T >, 1203
dds::pub::NoOpDataWriterListener< T >, 1842
dds::pub::NoOpPublisherListener, 1853
on_offered_incompatible_qos
dds::domain::NoOpDomainParticipantListener, 1847
dds::pub::AnyDataWriterListener, 822
dds::pub::DataWriterListener< T >, 1204
dds::pub::NoOpDataWriterListener< T >, 1842
dds::pub::NoOpPublisherListener, 1854
on_publication_matched
dds::domain::NoOpDomainParticipantListener, 1848
dds::pub::AnyDataWriterListener, 823
dds::pub::DataWriterListener< T >, 1205
dds::pub::NoOpDataWriterListener< T >, 1842
dds::pub::NoOpPublisherListener, 1854
on_reliable_reader_activity_changed
dds::domain::NoOpDomainParticipantListener, 1848
dds::pub::AnyDataWriterListener, 823
dds::pub::DataWriterListener< T >, 1206
dds::pub::NoOpDataWriterListener< T >, 1843
dds::pub::NoOpPublisherListener, 1855
on_reliable_writer_cache_changed
dds::domain::NoOpDomainParticipantListener, 1848
dds::pub::AnyDataWriterListener, 823
dds::pub::DataWriterListener< T >, 1205
dds::pub::NoOpDataWriterListener< T >, 1843
dds::pub::NoOpPublisherListener, 1854
on_reply_acknowledged
rti::queuing::NoOpQueueReplierListener< TReq, TRep >, 1860
rti::queuing::QueueReplierListener< TReq, TRep >, 2108
on_reply_available
rti::queuing::NoOpQueueRequesterListener< TReq, TRep >, 1862
rti::queuing::QueueRequesterListener< TReq, TRep >, 2127
on_reply_shared_reader_queue_matched
rti::queuing::NoOpQueueReplierListener< TReq, TRep >, 1861
rti::queuing::NoOpQueueRequesterListener< TReq, TRep >, 1863
rti::queuing::QueueReplierListener< TReq, TRep >, 2109
rti::queuing::QueueRequesterListener< TReq, TRep >, 2128
on_request_acknowledged
rti::queuing::NoOpQueueRequesterListener< TReq, TRep >, 1862

rti::queuing::QueueRequesterListener< TReq, TRep >, 2127
on_request_available
rti::queuing::NoOpQueueReplierListener< TReq, TRep >, 1860
rti::queuing::QueueReplierListener< TReq, TRep >, 2108
rti::request::ReplierListener< RequestType, ReplyType >, 2180
on_request_shared_reader_queue_matched
rti::queuing::NoOpQueueReplierListener< TReq, TRep >, 1860
rti::queuing::NoOpQueueRequesterListener< TReq, TRep >, 1863
rti::queuing::QueueReplierListener< TReq, TRep >, 2108
rti::queuing::QueueRequesterListener< TReq, TRep >, 2127
on_requested_deadline_missed
dds::domain::NoOpDomainParticipantListener, 1850
dds::sub::AnyDataReaderListener, 813
dds::sub::DataReaderListener< T >, 1057
dds::sub::NoOpDataReaderListener< T >, 1839
dds::sub::NoOpSubscriberListener, 1865
on_requested_incompatible_qos
dds::domain::NoOpDomainParticipantListener, 1850
dds::sub::AnyDataReaderListener, 813
dds::sub::DataReaderListener< T >, 1057
dds::sub::NoOpDataReaderListener< T >, 1839
dds::sub::NoOpSubscriberListener, 1865
on_sample_acknowledged
rti::queuing::NoOpQueueProducerListener< T >, 1858
rti::queuing::QueueProducerListener< T >, 2091
on_sample_available
rti::queuing::NoOpQueueConsumerListener< T >, 1857
rti::queuing::QueueConsumerListener< T >, 2076
on_sample_lost
dds::domain::NoOpDomainParticipantListener, 1851
dds::sub::AnyDataReaderListener, 814
dds::sub::DataReaderListener< T >, 1058
dds::sub::NoOpDataReaderListener< T >, 1840
dds::sub::NoOpSubscriberListener, 1866
on_sample_rejected
dds::domain::NoOpDomainParticipantListener, 1850
dds::sub::AnyDataReaderListener, 813
dds::sub::DataReaderListener< T >, 1057
dds::sub::NoOpDataReaderListener< T >, 1839
dds::sub::NoOpSubscriberListener, 1865
on_sample_removed
dds::domain::NoOpDomainParticipantListener, 1848
dds::pub::AnyDataWriterListener, 823
dds::pub::DataWriterListener< T >, 1209
dds::pub::NoOpDataWriterListener< T >, 1845
dds::pub::NoOpPublisherListener, 1855
on_service_request_accepted
dds::domain::NoOpDomainParticipantListener, 1849
dds::pub::AnyDataWriterListener, 824
dds::pub::DataWriterListener< T >, 1209
dds::pub::NoOpDataWriterListener< T >, 1844
dds::pub::NoOpPublisherListener, 1856
on_shared_reader_queue_matched
rti::queuing::NoOpQueueConsumerListener< T >, 1857
rti::queuing::NoOpQueueProducerListener< T >, 1858
rti::queuing::QueueConsumerListener< T >, 2077
rti::queuing::QueueProducerListener< T >, 2091
on_subscription_matched
dds::domain::NoOpDomainParticipantListener, 1851
dds::sub::AnyDataReaderListener, 814
dds::sub::DataReaderListener< T >, 1058
dds::sub::NoOpDataReaderListener< T >, 1840
dds::sub::NoOpSubscriberListener, 1866
on_thread_deleted
rti::core::cond::AsyncWaitSetListener, 857
rti::core::cond::NoOpAsyncWaitSetListener, 1837
on_thread_spawned
rti::core::cond::AsyncWaitSetListener, 857
rti::core::cond::NoOpAsyncWaitSetListener, 1837
on_wait_timeout
rti::core::cond::AsyncWaitSetListener, 857
rti::core::cond::NoOpAsyncWaitSetListener, 1837
operator bool
dds::core::external< T >, 1557
dds::core::optional< T >, 1885
rti::core::optional_value< T >, 1894
operator const D &
dds::core::Value< D >, 2603
operator const DataType &
rti::sub::LoanedSample< T >, 1662
operator const dds::core::string &
dds::core::StringTopicType, 2398
operator const DynamicData &
rti::core::xtypes::LoanedDynamicData, 1659
operator D&
dds::core::Value< D >, 2602
operator dds::core::InstanceHandle
dds::topic::TopicInstance< T >, 2506
operator dds::core::string &
dds::core::StringTopicType, 2398
operator DynamicData &
rti::core::xtypes::LoanedDynamicData, 1658
operator std::basic_string< CharType >
dds::core::basic_string< CharType, Allocator >, 879
operator std::string
dds::core::StringTopicType, 2397

operator std::vector< T >
dds::core::vector< T >, 2606
operator std::vector< uint8_t >
dds::core::BytesTopicType, 919
dds::core::KeyedBytesTopicType, 1627
operator!=
dds::core, 624–626
dds::core::basic_string< CharType, Allocator >, 878
dds::core::Duration, 1455
dds::core::external< T >, 1557
dds::core::optional< T >, 1890, 1891
dds::core::safe_enum< def, inner >, 2268
dds::core::Time, 2464
dds::core::vector< T >, 2610
dds::sub::status::DataState, 1122
rti::core::pointer< T >, 1940
rti::flat::OffsetBase, 1879
rti::flat::SequenceIterator< E, OffsetKind >, 2335
rti::sub::status::DataStateEx, 1130
operator<
dds::core::Duration, 1456
dds::core::safe_enum< def, inner >, 2268
dds::core::Time, 2465
rti::core::Guid, 1598
rti::core::SequenceNumber, 2341
rti::flat::OffsetBase, 1878
rti::flat::SequenceIterator< E, OffsetKind >, 2334
operator<<
dds::core, 622, 626
dds::core::basic_string< CharType, Allocator >, 879
dds::core::InstanceHandle, 1614
dds::core::optional< T >, 1891
dds::core::safe_enum< def, inner >, 2270
dds::core::TEntityQos< DELEGATE >, 2447
dds::core::vector< T >, 2611
dds::core::xtypes, 634
dds::core::xtypes::DynamicType, 1507
dds::domain::DomainParticipant, 1337
dds::domain::qos, 643, 645
dds::domain::qos::DomainParticipantFactoryQos, 1381
dds::domain::qos::DomainParticipantQos, 1388, 1390
dds::pub::DataWriter< T >, 1157, 1158, 1167
dds::pub::Publisher, 1998
dds::pub::qos, 656, 658
dds::pub::qos::DataWriterQos, 1232, 1234
dds::pub::qos::PublisherQos, 2009, 2013
dds::sub::DataReader< T >, 1008
dds::sub::qos, 684, 686
dds::sub::qos::DataReaderQos, 1078, 1080
dds::sub::qos::SubscriberQos, 2427
dds::sub::status, 687, 688
dds::sub::status::DataState, 1122, 1125
dds::sub::status::InstanceState, 1618
dds::sub::status::SampleState, 2318
dds::sub::status::ViewState, 2616
dds::topic::qos, 698
dds::topic::qos::TopicQos, 2512, 2515
rti::core, 708, 710
rti::core::CoherentSetInfo, 939
rti::core::Cookie, 967
rti::core::Guid, 1599
rti::core::SampleIdentity, 2284
rti::core::SequenceNumber, 2342
rti::core::status::EventCount< IntegerType >, 1545
rti::sub, 765
rti::sub::LoanedSample< T >, 1663
rti::sub::status::DataStateEx, 1129
operator<=
dds::core::Duration, 1455
dds::core::safe_enum< def, inner >, 2269
dds::core::Time, 2464
rti::core::Guid, 1598
rti::core::SequenceNumber, 2342
rti::flat::OffsetBase, 1878
rti::flat::SequenceIterator< E, OffsetKind >, 2334
operator>
dds::core::Duration, 1454
dds::core::safe_enum< def, inner >, 2269
dds::core::Time, 2462
rti::core::Guid, 1598
rti::core::SequenceNumber, 2342
rti::flat::OffsetBase, 1878
rti::flat::SequenceIterator< E, OffsetKind >, 2334
operator>>
dds::core::TEntityQos< DELEGATE >, 2448
dds::domain::DomainParticipant, 1337
dds::domain::qos::DomainParticipantQos, 1388
dds::pub::DataWriter< T >, 1167
dds::pub::Publisher, 1998
dds::pub::qos::DataWriterQos, 1232
dds::pub::qos::PublisherQos, 2010
dds::sub::DataReader< T >, 993, 995, 1009
dds::sub::DataReader< T >::ManipulatorSelector, 1699
dds::sub::qos::DataReaderQos, 1078
dds::sub::status::DataState, 1122, 1123
dds::topic::qos::TopicQos, 2513
rti::sub::status::DataStateEx, 1129
operator>=
dds::core::Duration, 1454
dds::core::safe_enum< def, inner >, 2269
dds::core::Time, 2462
rti::core::Guid, 1598
rti::core::SequenceNumber, 2342
rti::flat::OffsetBase, 1878
rti::flat::SequenceIterator< E, OffsetKind >, 2334

operator∗
dds::core, 620, 621
dds::core::Duration, 1459
dds::core::external< T >, 1555
dds::core::optional< T >, 1887
rti::core::optional_value< T >, 1896, 1897
rti::flat::SequenceIterator< E, OffsetKind >, 2332
operator()
rti::request::IsReplyRelatedPredicate< T >, 1623, 1624
rti::sub::IsValidData< T >, 1625
operator+
dds::core, 622, 623
dds::core::Duration, 1457
dds::core::Time, 2467, 2468
rti::core::SequenceNumber, 2340
operator++
rti::core::SequenceNumber, 2341
rti::flat::SequenceIterator< E, OffsetKind >, 2333
rti::sub::SampleIterator< T >, 2296
operator+=
dds::core::cond::WaitSet, 2624
dds::core::Duration, 1456
dds::core::Time, 2465
rti::core::cond::AsyncWaitSet, 851
rti::core::SequenceNumber, 2340
operator-
dds::core, 623, 624
dds::core::Duration, 1457
dds::core::Time, 2468, 2469
rti::core::SequenceNumber, 2340
operator->
dds::core::external< T >, 1556, 1557
dds::core::optional< T >, 1887, 1888
dds::core::Value< D >, 2601
rti::core::optional_value< T >, 1897
rti::flat::SequenceIterator< E, OffsetKind >, 2333
operator--
rti::core::SequenceNumber, 2341
operator-=
dds::core::cond::WaitSet, 2624
dds::core::Duration, 1456
dds::core::Time, 2466
rti::core::cond::AsyncWaitSet, 851
rti::core::SequenceNumber, 2340
operator/
dds::core, 621
dds::core::Duration, 1460
operator=
dds::core::basic_string< CharType, Allocator >, 878
dds::core::external< T >, 1555
dds::core::optional< T >, 1888, 1889
dds::core::TEntityQos< DELEGATE >, 2448
dds::core::vector< T >, 2609
dds::pub::AnyDataWriter, 817, 819
dds::pub::qos::DataWriterQos, 1230
dds::sub::AnyDataReader, 809, 810
dds::sub::qos::DataReaderQos, 1076
dds::sub::Sample< T >, 2272, 2273
rti::core::bounded_sequence< T, MaxLength >, 895, 896
rti::core::pointer< T >, 1940
rti::core::xtypes::LoanedDynamicData, 1659
operator==
dds::core, 624, 625
dds::core::basic_string< CharType, Allocator >, 878
dds::core::Duration, 1454
dds::core::external< T >, 1557
dds::core::InstanceHandle, 1613
dds::core::optional< T >, 1889, 1890
dds::core::safe_enum< def, inner >, 2268
dds::core::Time, 2464
dds::core::vector< T >, 2609
dds::sub::status::DataState, 1121
rti::core::pointer< T >, 1940
rti::flat::OffsetBase, 1879
rti::flat::SequenceIterator< E, OffsetKind >, 2335
rti::sub, 767
rti::sub::LoanedSample< T >, 1662
rti::sub::status::DataStateEx, 1130
operator[]
dds::core::BytesTopicType, 920
dds::core::KeyedBytesTopicType, 1628
dds::core::vector< T >, 2609
dds::sub::LoanedSamples< T >, 1666
dds::sub::SharedSamples< T, DELEGATE >, 2364
rti::core::bounded_sequence< T, MaxLength >, 897
rti::core::Guid, 1597, 1598
rti::core::LongDouble, 1697
optional
dds::core::optional< T >, 1882, 1883
dds::core::xtypes::Member, 1705
Supporting Types and Constants, 436
optional_value
rti::core::optional_value< T >, 1893, 1894
ordered_access
dds::core::policy::Presentation, 1948
ordinal
dds::core::xtypes::EnumMember, 1534
original_publication_virtual_guid
dds::sub::SampleInfo, 2291
original_publication_virtual_sample_identity
dds::sub::SampleInfo, 2292
original_publication_virtual_sequence_number
dds::sub::SampleInfo, 2291
original_related_reader_guid
rti::sub::TopicQueryData, 2521
OsapiFile, 1898

Other QoS Attributes, 286
Other Utilities, 595
enable_automatic_tss_cleanup, 596
sleep, 596
spin, 597
spin_per_microsecond, 597
unload_activation_library, 598
out
rti::util::network_capture::TrafficKindMask, 2531
out_of_range_rejected_sample_count
rti::core::status::DataReaderProtocolStatus, 1072
output_file
rti::config::Logger, 1686, 1687
output_file_set
rti::config::Logger, 1687
output_handler
rti::config::Logger, 1687
outstanding_asynchronous_sample_allocation
rti::core::policy::DomainParticipantResourceLimits, 1405
OWNERSHIP, 538
OwnershipKind, 539
Ownership
dds::core::policy::Ownership, 1904
ownership
dds::topic::PublicationBuiltinTopicData, 1982
dds::topic::SubscriptionBuiltinTopicData, 2432
dds::topic::TopicBuiltinTopicData, 2499
ownership_dropped_sample_count
rti::core::status::DataReaderCacheStatus, 1049
OWNERSHIP_STRENGTH, 539
ownership_strength
dds::topic::PublicationBuiltinTopicData, 1983
OwnershipKind
OWNERSHIP, 539
OwnershipStrength
dds::core::policy::OwnershipStrength, 1907

parameter_length_with_padding
rti::config::compliance::XTypesMask, 2669
parameters
dds::sub::cond::QueryCondition, 2062, 2063
dds::sub::Query, 2057, 2058
dds::topic::Filter, 1561
parameters_length
dds::sub::cond::QueryCondition, 2063
dds::sub::Query, 2058
dds::topic::Filter, 1562
parent
dds::core::xtypes::StructType, 2408
NDDS_Transport_Shmem_Property_t, 1795
NDDS_Transport_UDPv4_Property_t, 1800
NDDS_Transport_UDPv4_WAN_Property_t, 1810
NDDS_Transport_UDPv6_Property_t, 1820
parse_encrypted_content
rti::util::network_capture::NetworkCaptureParams, 1832
partial_configuration
dds::topic::ParticipantBuiltinTopicData, 1915
participant
dds::pub::Publisher, 2001
dds::sub::Subscriber, 2418
dds::topic::TopicDescription< T >, 2504
rti::pub::FlowController, 1574
Participant Built-in Topics, 441
participant_topic_name, 441
Participant Use Cases, 306
participant_announcement_period
rti::core::policy::DiscoveryConfig, 1280
participant_configuration_reader
rti::core::policy::DiscoveryConfig, 1312, 1313
participant_configuration_reader_resource_limits
rti::core::policy::DiscoveryConfig, 1313, 1314
participant_configuration_writer
rti::core::policy::DiscoveryConfig, 1314, 1315
participant_configuration_writer_data_lifecycle
rti::core::policy::DiscoveryConfig, 1316
participant_configuration_writer_publish_mode
rti::core::policy::DiscoveryConfig, 1316, 1317
participant_factory_qos
dds::core::QosProvider, 2037
dds::domain::DomainParticipant, 1340
participant_id
rti::core::policy::WireProtocol, 2635
participant_id_gain
rti::core::RtpsWellKnownPorts, 2263
participant_key
dds::topic::PublicationBuiltinTopicData, 1980
dds::topic::SubscriptionBuiltinTopicData, 2430
participant_liveliness_assert_period
rti::core::policy::DiscoveryConfig, 1279, 1280
participant_liveliness_lease_duration
rti::core::policy::DiscoveryConfig, 1279
participant_message_reader
rti::core::policy::DiscoveryConfig, 1295, 1296
participant_message_reader_reliability_kind
rti::core::policy::DiscoveryConfig, 1295
participant_message_writer
rti::core::policy::DiscoveryConfig, 1296, 1297
participant_name
dds::topic::ParticipantBuiltinTopicData, 1914
rti::domain::DomainParticipantConfigParams, 1374
participant_property_list_max_length
rti::core::policy::DomainParticipantResourceLimits, 1424
participant_property_string_max_length
rti::core::policy::DomainParticipantResourceLimits, 1425

participant_protocol_status
dds::domain::DomainParticipant, 1358
participant_qos
dds::core::QosProvider, 2031
participant_qos_library_name
rti::domain::DomainParticipantConfigParams, 1374
participant_qos_profile_name
rti::core::MonitoringDedicatedParticipantSettings, 1714, 1715
rti::domain::DomainParticipantConfigParams, 1375
participant_reader_resource_limits
rti::core::policy::DiscoveryConfig, 1285, 1286
participant_topic_name
Participant Built-in Topics, 441
participant_user_data_max_length
rti::core::policy::DomainParticipantResourceLimits, 1414
ParticipantTrustAlgorithmInfo
rti::topic::trust::ParticipantTrustAlgorithmInfo, 1916
ParticipantTrustInterceptorAlgorithmInfo
rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo, 1918
ParticipantTrustKeyEstablishmentAlgorithmInfo
rti::topic::trust::ParticipantTrustKeyEstablishmentAlgorithmInfo, 1920
ParticipantTrustProtectionInfo
rti::topic::trust::ParticipantTrustProtectionInfo, 1921
ParticipantTrustSignatureAlgorithmInfo
rti::topic::trust::ParticipantTrustSignatureAlgorithmInfo, 1922
PARTITION, 539
partition
dds::topic::ParticipantBuiltinTopicData, 1912
dds::topic::PublicationBuiltinTopicData, 1983
dds::topic::SubscriptionBuiltinTopicData, 2433
pattern_alarm_event
Builtin Qos Profiles, 481, 490
pattern_alarm_status
Builtin Qos Profiles, 482, 490
pattern_event
Builtin Qos Profiles, 481, 489
pattern_last_value_cache
Builtin Qos Profiles, 482, 491
pattern_periodic_data
Builtin Qos Profiles, 480, 488
pattern_reliable_streaming
Builtin Qos Profiles, 481, 489
pattern_rpc
Builtin Qos Profiles, 480
pattern_status
Builtin Qos Profiles, 482, 490
pattern_streaming
Builtin Qos Profiles, 480, 489
pause
Heap Monitoring, 592
Network Capture, 587
period
dds::core::policy::Deadline, 1258
rti::pub::FlowControllerTokenBucketProperty, 1587
periodic_settings
rti::core::MonitoringDistributionSettings, 1720, 1721
persist_journal
DURABILITY, 530
PERSISTENCE
rti::core::policy::ServiceKind_def, 2358
PersistenceService
rti::core::policy::Service, 2352
PERSISTENT
dds::core::policy::DurabilityKind_def, 1442
Persistent
dds::core::policy::Durability, 1438
PersistentJournalKind
DURABILITY, 530
PersistentStorageSettings
rti::core::PersistentStorageSettings, 1929
PersistentSynchronizationKind
DURABILITY, 530
plain_cast
FlatData Offsets, 417, 418
platform
rti::config::LogCategory_def, 1682
plugin_bitmask
rti::topic::trust::EndpointTrustProtectionInfo, 1520
rti::topic::trust::ParticipantTrustProtectionInfo, 1921
plugin_data
rti::core::policy::TypeSupport, 2575
pointer
dds::core::xtypes::Member, 1705
dds::core::xtypes::UnionMember, 2583
rti::core::bounded_sequence< T, MaxLength >, 892
rti::core::pointer< T >, 1939
rti::flat::SequenceIterator< E, OffsetKind >, 2331
policies
dds::core::status::OfferedIncompatibleQosStatus, 1874
dds::core::status::RequestedIncompatibleQosStatus, 2186
policy
dds::core::TEntityQos< DELEGATE >, 2447
dds::domain::qos::DomainParticipantFactoryQos, 1379
dds::domain::qos::DomainParticipantQos, 1387
dds::pub::qos::DataWriterQos, 1231
dds::pub::qos::PublisherQos, 2008, 2009
dds::sub::qos::DataReaderQos, 1077
dds::sub::qos::SubscriberQos, 2425
dds::topic::qos::TopicQos, 2511, 2512
policy_id

dds::core::policy::QosPolicyCount, 2021
polling_period
rti::core::MonitoringPeriodicDistributionSettings, 1744
polymorphic_cast
dds::core, 620
dds::core::Entity, 1526
pop_back
rti::core::bounded_sequence< T, MaxLength >, 904
port
rti::core::Locator, 1674, 1675
port_base
rti::core::RtpsWellKnownPorts, 2262
port_offset
NDDS_Transport_UDPv4_Property_t, 1808
NDDS_Transport_UDPv4_WAN_Property_t, 1818
NDDS_Transport_UDPv6_Property_t, 1828
PRESENTATION, 540
PresentationAccessScopeKind, 540
Presentation
dds::core::policy::Presentation, 1946
presentation
dds::topic::PublicationBuiltinTopicData, 1983
dds::topic::SubscriptionBuiltinTopicData, 2433
PresentationAccessScopeKind
PRESENTATION, 540
pretty_print
rti::topic::PrintFormatProperty, 1964
prevent_type_widening
dds::core::policy::TypeConsistencyEnforcement, 2568
PRIMITIVE_TYPE
dds::core::xtypes::TypeKind_def, 2572
primitive_type
dds::core::xtypes, 634
dds::core::xtypes::PrimitiveType, 1960
print_complete_type
rti::core::xtypes::DynamicTypePrintFormatProperty, 1513
print_format
rti::config::Logger, 1688
print_format_by_log_level
rti::config::Logger, 1688, 1689
print_idl
dds::core::xtypes::DynamicType, 1508
rti::core::xtypes, 725
print_kind
rti::core::xtypes::DynamicTypePrintFormatProperty, 1512, 1513
print_ordinals
rti::core::xtypes::DynamicTypePrintFormatProperty, 1511
print_private
rti::core::QosPrintFormat, 2023
PrintFormat
Logging, 457
PrintFormatKind
Topic traits and data-type support, 445
PrintFormatProperty
rti::topic::PrintFormatProperty, 1963
priority
rti::core::ChannelSettings, 924
rti::core::policy::PublishMode, 2017, 2018
rti::core::ThreadSettings, 2450, 2451
rti::pub::WriteParams, 2648
rti::topic::FilterSampleInfo, 1565
priority_enforce
rti::core::ThreadSettingsKindMask, 2456
producer
rti::queuing::QueueReplier< TReq, TRep >, 2102
rti::queuing::QueueRequester< TReq, TRep >, 2122
product_version
dds::topic::ParticipantBuiltinTopicData, 1913
dds::topic::PublicationBuiltinTopicData, 1988
dds::topic::SubscriptionBuiltinTopicData, 2437
Version, 461
ProductVersion
rti::core::ProductVersion, 1967
PROFILE, 541
profiles_loaded
dds::core::QosProvider, 2043
Programming How-To's, 362
propagate
rti::core::policy::Property, 1975
propagate_app_ack_with_no_response
rti::core::policy::DataWriterProtocol, 1216, 1217
propagate_dispose_of_unregistered_instances
rti::core::policy::DataReaderProtocol, 1063
propagate_unregister_of_disposed_instances
rti::core::policy::DataReaderProtocol, 1064
properties_bitmap
NDDS_Transport_Property_t, 1788
PROPERTY, 541
Property
rti::core::policy::Property, 1972
property
dds::core::cond::WaitSet, 2627
dds::domain::DomainParticipant, 1338
dds::topic::ParticipantBuiltinTopicData, 1910
dds::topic::PublicationBuiltinTopicData, 1987
dds::topic::SubscriptionBuiltinTopicData, 2435
rti::core::cond::AsyncWaitSet, 852
rti::pub::FlowController, 1574
PROTOCOL
rti::core::policy::AcknowledgmentKind_def, 797
protocol_overhead_max
NDDS_Transport_UDPv4_Property_t, 1806
NDDS_Transport_UDPv4_WAN_Property_t, 1815

NDDS_Transport_UDPv6_Property_t, 1826
ProtocolVersion
rti::core::ProtocolVersion, 1976
provider_params
dds::core::QosProvider, 2044, 2045
public_address
NDDS_Transport_UDPv4_Property_t, 1807
NDDS_Transport_UDPv4_WAN_Property_t, 1816
NDDS_Transport_UDPv6_Property_t, 1827
public_port
NDDS_Transport_UDP_WAN_CommPortsMappingInfo, 1798
Publication Built-in Topics, 442
publication_topic_name, 443
Publication Example, 304
Publication Module, 55
publication_handle
dds::sub::SampleInfo, 2290
publication_matched
dds::core::status::StatusMask, 2384
publication_matched_status
dds::pub::DataWriter< T >, 1172
publication_name
dds::topic::PublicationBuiltinTopicData, 1989
publication_period
rti::core::MonitoringEventDistributionSettings, 1726, 1727
rti::core::MonitoringLoggingDistributionSettings, 1732
rti::core::policy::TopicQueryDispatch, 2525
PUBLICATION_PRIORITY_AUTOMATIC
PUBLISH_MODE, 543
PUBLICATION_PRIORITY_UNDEFINED
PUBLISH_MODE, 542
publication_reader
rti::core::policy::DiscoveryConfig, 1286, 1287
publication_reader_resource_limits
rti::core::policy::DiscoveryConfig, 1287
publication_sequence_number
dds::sub::SampleInfo, 2291
publication_topic_name
Publication Built-in Topics, 443
publication_writer
rti::core::policy::DiscoveryConfig, 1289, 1290
publication_writer_data_lifecycle
rti::core::policy::DiscoveryConfig, 1291
publication_writer_publish_mode
rti::core::policy::DiscoveryConfig, 1298
PUBLISH_MODE, 541
PUBLICATION_PRIORITY_AUTOMATIC, 543
PUBLICATION_PRIORITY_UNDEFINED, 542
PublishModeKind, 542
Publisher
dds::pub::Publisher, 1995, 1996
publisher
dds::pub::AnyDataWriter, 818
dds::pub::DataWriter< T >, 1168
rti::request::ReplierParams, 2182
rti::request::RequesterParams, 2201
Publisher Use Cases, 310
publisher_group_data_max_length
rti::core::policy::DomainParticipantResourceLimits, 1415
publisher_key
dds::topic::PublicationBuiltinTopicData, 1986
publisher_qos
dds::core::QosProvider, 2034
publisher_qos_profile_name
rti::core::MonitoringDistributionSettings, 1718, 1719
Publishers, 55
PublishMode
rti::core::policy::PublishMode, 2015
PublishModeKind
PUBLISH_MODE, 542
pulled_fragment_bytes
rti::core::status::DataWriterProtocolStatus, 1226
pulled_fragment_count
rti::core::status::DataWriterProtocolStatus, 1225
pulled_sample_bytes
rti::core::status::DataWriterProtocolStatus, 1222
pulled_sample_count
rti::core::status::DataWriterProtocolStatus, 1221
push_back
rti::core::bounded_sequence< T, MaxLength >, 903, 904
push_on_write
rti::core::policy::DataWriterProtocol, 1213
pushed_fragment_bytes
rti::core::status::DataWriterProtocolStatus, 1225
pushed_fragment_count
rti::core::status::DataWriterProtocolStatus, 1225
pushed_sample_bytes
rti::core::status::DataWriterProtocolStatus, 1220
pushed_sample_count
rti::core::status::DataWriterProtocolStatus, 1220
python_int8_730
rti::config::compliance::XTypesMask, 2673

QoS, 508
qos
dds::domain::DomainParticipant, 1336, 1337
dds::pub::AnyDataWriter, 817
dds::pub::DataWriter< T >, 1166
dds::pub::Publisher, 1997
dds::sub::AnyDataReader, 809
dds::sub::DataReader< T >, 1007, 1008
dds::sub::Subscriber, 2417
dds::topic::AnyTopic, 826, 827

dds::topic::Topic< T >, 2483, 2484
QoS by Category, 514
QoS Mutability, 242
QoS Policy Traits, 428
QoS Profile Management, 220
Qos Provider Use Cases, 604
Qos Use Cases, 602
QOS_ELEMENT_NAME_USE_XML_CONFIG
rti::domain::DomainParticipantConfigParams, 1377
qos_print_all
Supporting Types and Constants, 439
qos_profile
rti::queuing::QueueEntityParams< ActualEntity >, 2079
qos_profile_libraries
dds::core::QosProvider, 2043
qos_profiles
dds::core::QosProvider, 2043
QosPolicyCount
dds::core::policy::QosPolicyCount, 2020
QosPolicyCountSeq
All QoS Policies, 514
QosPolicyId
All QoS Policies, 514
QosPrintFormat
rti::core::QosPrintFormat, 2022
QosProvider
dds::core::QosProvider, 2030
QosProviderParams
rti::core::QosProviderParams, 2050
Queries and Filters Syntax, 84
Query
dds::sub::Query, 2055, 2056
Query Conditions, 65
create_query_condition_ex, 66
query_condition_allocation
rti::core::policy::DomainParticipantResourceLimits, 1405
QueryCondition
dds::sub::cond::QueryCondition, 2061
QueueConsumer, 95
rti::queuing::QueueConsumer< T >, 2066, 2067
QueueConsumerParams
rti::queuing::QueueConsumerParams, 2078
QueueProducer, 95
rti::queuing::QueueProducer< T >, 2082, 2083
QueueProducerParams
rti::queuing::QueueProducerParams, 2093
QueueReplier, 96
rti::queuing::QueueReplier< TReq, TRep >, 2096
QueueReplierParams
rti::queuing::QueueReplierParams, 2110
QueueRequester, 96
rti::queuing::QueueRequester< TReq, TRep >, 2114
QueueRequesterParams
rti::queuing::QueueRequesterParams, 2129
QUEUING
rti::core::policy::ServiceKind_def, 2358
Queuing Pattern, 93
QueuingService
rti::core::policy::Service, 2352
quorum_count
rti::core::EndpointGroup, 1516

Rank
dds::sub::Rank, 2131
rank
dds::sub::SampleInfo, 2289
NDDS_Transport_Interface_t, 1786
rbegin
rti::core::bounded_sequence< T, MaxLength >, 900
reachability_lease_duration
dds::topic::ParticipantBuiltinTopicData, 1915
read
dds::sub, 661
dds::sub::DataReader< T >, 996, 1000, 1001, 1018, 1019, 1024
dds::sub::DataReader< T >::Selector, 2325–2327
dds::sub::status::SampleState, 2318
Read Conditions, 65
read_condition_allocation
rti::core::policy::DomainParticipantResourceLimits, 1404
read_noexcept
dds::sub::DataReader< T >, 1018
read_replies
rti::queuing::QueueRequester< TReq, TRep >, 2118, 2119
rti::request::Requester< RequestType, ReplyType >, 2194, 2195
read_requests
rti::queuing::QueueReplier< TReq, TRep >, 2105, 2106
rti::request::Replier< RequestType, ReplyType >, 2176
read_samples
rti::queuing::QueueConsumer< T >, 2073
ReadCondition
dds::sub::cond::ReadCondition, 2134
reader
dds::sub::DataReader< T >::Selector, 2326
rti::queuing::QueueConsumer< T >, 2070
rti::queuing::QueueReplier< TReq, TRep >, 2106
rti::queuing::QueueRequester< TReq, TRep >, 2125
reader_checkpoint_frequency
rti::core::PersistentStorageSettings, 1937, 1938
READER_DATA_LIFECYCLE, 543
reader_data_tag_list_max_length

rti::core::policy::DomainParticipantResourceLimits, 1432
reader_data_tag_string_max_length
rti::core::policy::DomainParticipantResourceLimits, 1432
reader_property_list_max_length
rti::core::policy::DomainParticipantResourceLimits, 1426
reader_property_string_max_length
rti::core::policy::DomainParticipantResourceLimits, 1427
reader_user_data_max_length
rti::core::policy::DomainParticipantResourceLimits, 1416
ReaderDataLifecycle
dds::core::policy::ReaderDataLifecycle, 2139, 2140
readers
rti::sub::SampleProcessor, 2310
Real-Time WAN Transport, 394
NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT, 398
NDDS_Transport_UDPv4_WAN_new, 398
NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT, 398
realtime_priority
rti::core::ThreadSettingsKindMask, 2456
reassembled_sample_count
rti::core::status::DataReaderProtocolStatus, 1071
receive_address
rti::core::TransportMulticastSettings, 2551
receive_buffer_size
NDDS_Transport_Shmem_Property_t, 1795
receive_port
rti::core::TransportMulticastSettings, 2551
rti::core::TransportUnicastSettings, 2561
receive_replies
rti::queuing::QueueRequester< TReq, TRep >, 2116, 2117
rti::request::Requester< RequestType, ReplyType >, 2192, 2193
receive_requests
rti::queuing::QueueReplier< TReq, TRep >, 2103, 2104
rti::request::Replier< RequestType, ReplyType >, 2175
receive_samples
rti::queuing::QueueConsumer< T >, 2071
receive_window_size
rti::core::RtpsReliableReaderProtocol, 2226
received_ack_bytes
rti::core::status::DataWriterProtocolStatus, 1222
received_ack_count
rti::core::status::DataWriterProtocolStatus, 1222
received_fragment_count
rti::core::status::DataReaderProtocolStatus, 1071
received_gap_bytes
rti::core::status::DataReaderProtocolStatus, 1069
received_gap_count
rti::core::status::DataReaderProtocolStatus, 1069
received_heartbeat_bytes
rti::core::status::DataReaderProtocolStatus, 1068
received_heartbeat_count
rti::core::status::DataReaderProtocolStatus, 1068
received_message_count_max
NDDS_Transport_Shmem_Property_t, 1795
received_nack_bytes
rti::core::status::DataWriterProtocolStatus, 1222
received_nack_count
rti::core::status::DataWriterProtocolStatus, 1222
received_nack_fragment_bytes
rti::core::status::DataWriterProtocolStatus, 1226
received_nack_fragment_count
rti::core::status::DataWriterProtocolStatus, 1226
received_sample_bytes
rti::core::status::DataReaderProtocolStatus, 1067
received_sample_count
rti::core::status::DataReaderProtocolStatus, 1066
RECEIVER_POOL, 543
ReceiverPool
rti::core::policy::ReceiverPool, 2145
reception_enabled
rti::queuing::ConsumerAvailabilityParams, 950
reception_sequence_number
dds::sub::SampleInfo, 2291
reception_timestamp
dds::sub::SampleInfo, 2290
ReceptionTimestamp
dds::core::policy::DestinationOrder, 1262
RECORDING
rti::core::policy::ServiceKind_def, 2358
RecordingService
rti::core::policy::Service, 2353
recover_state
RELIABILITY, 545
recv_socket_buffer_size
NDDS_Transport_UDPv4_Property_t, 1800
NDDS_Transport_UDPv4_WAN_Property_t, 1810
NDDS_Transport_UDPv6_Property_t, 1821
redelivered
rti::core::SampleFlag, 2280
reference
rti::core::bounded_sequence< T, MaxLength >, 891
rti::flat::SequenceIterator< E, OffsetKind >, 2331
register_contentfilter
dds::domain::DomainParticipant, 1349
register_durable_subscription
dds::domain::DomainParticipant, 1358
register_instance

dds::pub::DataWriter< T >, 1158, 1159, 1185
register_type
dds::domain::DomainParticipant, 1370
rti::domain, 733, 734
rejected_by_decode_failure
dds::core::status::SampleRejectedState, 2314
rejected_by_instances_limit
dds::core::status::SampleRejectedState, 2313
rejected_by_remote_writers_per_virtual_queue_limit
dds::core::status::SampleRejectedState, 2314
rejected_by_samples_limit
dds::core::status::SampleRejectedState, 2312
rejected_by_samples_per_instance_limit
dds::core::status::SampleRejectedState, 2313
rejected_by_samples_per_remote_writer_limit
dds::core::status::SampleRejectedState, 2313
rejected_sample_count
rti::core::status::DataReaderProtocolStatus, 1070
rti::core::status::DataWriterProtocolStatus, 1223
related_datareader_key
dds::topic::PublicationBuiltinTopicData, 1990
related_datawriter_key
dds::topic::SubscriptionBuiltinTopicData, 2438
related_original_publication_virtual_guid
dds::sub::SampleInfo, 2292
related_original_publication_virtual_sample_identity
dds::sub::SampleInfo, 2293
related_original_publication_virtual_sequence_number
dds::sub::SampleInfo, 2292
related_reader_guid
rti::pub::WriteParams, 2651
rti::topic::FilterSampleInfo, 1564
related_sample_identity
rti::pub::WriteParams, 2646
rti::topic::FilterSampleInfo, 1563
related_source_guid
dds::sub::SampleInfo, 2294
rti::pub::WriteParams, 2651
rti::topic::FilterSampleInfo, 1563
related_subscription_guid
dds::sub::SampleInfo, 2294
related_topic_name
rti::core::ContentFilterProperty, 961
related_type
dds::core::xtypes::AliasType, 802
release_version
rti::config::LibraryVersion, 1634
rti::core::ProductVersion, 1968
RELIABILITY, 544
AcknowledgmentKind, 545
InstanceStateConsistencyKind, 545
none, 545
recover_state, 545
ReliabilityKind, 545
Reliability
dds::core::policy::Reliability, 2152
reliability
dds::topic::PublicationBuiltinTopicData, 1982
dds::topic::SubscriptionBuiltinTopicData, 2432
dds::topic::TopicBuiltinTopicData, 2497
Reliability QoS, 222
ReliabilityKind
RELIABILITY, 545
RELIABLE
dds::core::policy::ReliabilityKind_def, 2157
Reliable
dds::core::policy::Reliability, 2154
reliable_reader_activity_changed
dds::core::status::StatusMask, 2385
reliable_reader_activity_changed_status
dds::pub::DataWriter< T >, 1175
reliable_writer_cache_changed
dds::core::status::StatusMask, 2385
reliable_writer_cache_changed_status
dds::pub::DataWriter< T >, 1175
reload_profiles
dds::core::QosProvider, 2045
Remote Procedure Call, 97
remote_participant_allocation
rti::core::policy::DomainParticipantResourceLimits, 1401
remote_participant_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1410
remote_participant_purge_kind
rti::core::policy::DiscoveryConfig, 1281
remote_reader_allocation
rti::core::policy::DomainParticipantResourceLimits, 1401
remote_reader_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1409
remote_topic_query_allocation
rti::core::policy::DomainParticipantResourceLimits, 1429, 1430
remote_topic_query_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1430
remote_writer_allocation
rti::core::policy::DomainParticipantResourceLimits, 1400
remote_writer_hash_buckets
rti::core::policy::DomainParticipantResourceLimits, 1409
RemoteParticipantPurgeKind
DISCOVERY_CONFIG, 526
remove
dds::core::policy::DataTag, 1137

rti::core::policy::Property, 1974
remove_from_expression_parameter
dds::topic::ContentFilteredTopic< T >, 959
remove_peer
dds::domain::DomainParticipant, 1353
rend
rti::core::bounded_sequence< T, MaxLength >, 901
replace_automatic_values
rti::pub::WriteParams, 2644
replace_empty_instances
rti::core::policy::DataWriterResourceLimits, 1242
replaced_dropped_sample_count
rti::core::status::DataReaderCacheStatus, 1050
replaced_unacknowledged_sample_count
rti::core::status::ReliableWriterCacheChangedStatus, 2161
REPLAY
rti::core::policy::ServiceKind_def, 2358
ReplayService
rti::core::policy::Service, 2353
replicate
rti::core::SampleFlag, 2280
Replier, 93
rti::request::Replier< RequestType, ReplyType >, 2171, 2172
ReplierParams
rti::request::ReplierParams, 2181
reply_datareader
dds::rpc::ClientEndpoint< Request, Reply >, 927
rti::request::Requester< RequestType, ReplyType >, 2197
reply_datawriter
dds::rpc::ServiceEndpoint< Dispatcher >, 2357
rti::request::Replier< RequestType, ReplyType >, 2178
reply_topic_name
rti::request::ReplierParams, 2182
rti::request::RequesterParams, 2201
reply_type
rti::request::ReplierParams, 2183
rti::request::RequesterParams, 2202
ReplyType
dds::rpc::ClientEndpoint< Request, Reply >, 926
dds::rpc::ServiceEndpoint< Dispatcher >, 2356
representation
dds::topic::PublicationBuiltinTopicData, 1984, 1985
dds::topic::SubscriptionBuiltinTopicData, 2434
dds::topic::TopicBuiltinTopicData, 2500
Request-Reply Examples, 344
Request-Reply Pattern, 91
request_body
rti::topic::ServiceRequest, 2359
request_datareader
dds::rpc::ServiceEndpoint< Dispatcher >, 2357
rti::request::Replier< RequestType, ReplyType >, 2178
request_datawriter
dds::rpc::ClientEndpoint< Request, Reply >, 927
rti::request::Requester< RequestType, ReplyType >, 2197
request_reply_api_version
Version, 460
request_reply_build_number
Version, 460
request_topic_name
rti::request::ReplierParams, 2182
rti::request::RequesterParams, 2200
request_type
rti::request::ReplierParams, 2183
rti::request::RequesterParams, 2202
request_types_filter
rti::core::policy::DiscoveryConfig, 1317, 1318
requested_deadline_missed
dds::core::status::StatusMask, 2380
requested_deadline_missed_status
dds::sub::DataReader< T >, 1011
requested_incompatible_qos
dds::core::status::StatusMask, 2381
requested_incompatible_qos_status
dds::sub::DataReader< T >, 1012
Requester, 92
rti::request::Requester< RequestType, ReplyType >, 2189, 2190
RequesterParams
rti::request::RequesterParams, 2200
RequestType
dds::rpc::ClientEndpoint< Request, Reply >, 926
dds::rpc::ServiceEndpoint< Dispatcher >, 2355
require_matching_service_on_send_request
rti::request::RequesterParams, 2202
required_mask
rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo, 1518
rti::topic::trust::TrustAlgorithmRequirements, 2563
required_matched_endpoint_groups
rti::core::policy::Availability, 873, 874
reserve
dds::core::vector< T >, 2608
rti::core::bounded_sequence< T, MaxLength >, 902
RESERVED_RANGE
rti::core::TransportClassId_def, 2541
reset
dds::core::optional< T >, 1885
rti::core::optional_value< T >, 1895
rti::pub::WriteParams, 2644
reset_default
dds::core::QosProvider, 2036
reset_handler

dds::core::cond::GuardCondition, 1595
dds::core::cond::StatusCondition, 2375
reset_output_handler
rti::config::Logger, 1688
resize
dds::core::vector< T >, 2607
rti::core::bounded_sequence< T, MaxLength >, 904, 905
resolve_alias
dds::core::xtypes::AliasType, 802
rti::core::xtypes, 721
RESOURCE_LIMITS, 546
resource_limits
dds::topic::TopicBuiltinTopicData, 2499
resource_selection
rti::core::MonitoringMetricSelection, 1737, 1738
ResourceLimits
dds::core::policy::ResourceLimits, 2205
response_data
rti::pub::AcknowledgmentInfo, 796
restore
rti::core::PersistentStorageSettings, 1934
Result
rti::core::Result< T >, 2210, 2211
resume
dds::pub::SuspendedPublication, 2443
Heap Monitoring, 593
Network Capture, 588
resume_endpoint_discovery
dds::domain::DomainParticipant, 1356
retain
dds::core::Entity, 1526
dds::pub::AnyDataWriter, 819
dds::sub::AnyDataReader, 811
rti::pub::FlowController, 1576
return_loan
dds::sub::LoanedSamples< T >, 1667
rti::core::xtypes::LoanedDynamicData, 1658
return_loan_noexcept
dds::sub::LoanedSamples< T >, 1667
reuse_multicast_receive_resource
NDDS_Transport_UDPv4_Property_t, 1806
NDDS_Transport_UDPv6_Property_t, 1826
reverse_iterator
rti::core::bounded_sequence< T, MaxLength >, 893
revision_version
rti::core::ProductVersion, 1968
RobotControlClientEndpoint
Client-side API, 407
RobotControlService
Server-side API, 408
role_name
rti::core::EndpointGroup, 1515, 1516
rti::core::policy::EntityName, 1532
root
rti::flat::Sample< OffsetType >, 2276
ROUND_ROBIN
rti::core::ThreadSettingsCpuRotationKind_def, 2454
rti::pub::FlowControllerSchedulingPolicy_def, 1582
round_trip_time
rti::core::RtpsReliableReaderProtocol, 2226
RoundRobin
rti::pub::FlowControllerProperty, 1580
ROUTING
rti::core::policy::ServiceKind_def, 2358
RoutingService
rti::core::policy::Service, 2352
RPC Tutorial, 421
rpc_example::RobotControl, 2213
~RobotControl, 2214
get_speed, 2215
walk_to, 2214
rpc_example::RobotControlAsync, 2215
~RobotControlAsync, 2216
get_speed_async, 2217
walk_to_async, 2216
rpc_example::RobotControlClient, 2217
RpcMask
rti::config::compliance::RpcMask, 2219, 2220
rti, 698
RTI Connext DDS API Reference, 357
rti::all, 699
rti::config, 699
rti::config::activity_context::AttributeKindMask, 863
all, 867
AttributeKindMask, 864
default_mask, 867
domain_id, 866
entity_kind, 866
entity_name, 866
guid_prefix, 865
MaskType, 864
none, 867
topic, 865
type, 865
rti::config::compliance::RpcMask, 2218
default_mask, 2221
MaskType, 2219
RpcMask, 2219, 2220
use_standard_related_sample_pid, 2220
vendor, 2221
rti::config::compliance::XTypesMask, 2666
accept_unknown_discriminator, 2671
accept_unknown_enum_value, 2670
default_mask, 2674
dheader_in_non_primitive_collections, 2669
encapsulation_options_with_padding, 2670
enum_as_primitive_in_collections, 2669

initialize_discriminator_to_default, 2672
java_keyhash_750, 2672
MaskType, 2668
must_understand_in_key_and_discriminator, 2673
parameter_length_with_padding, 2669
python_int8_730, 2673
select_default_discriminator, 2671
sentinel_in_empty_union, 2671
vendor, 2674
XTypesMask, 2668
rti::config::LibraryVersion, 1634
build_version, 1635
major_version, 1634
minor_version, 1634
release_version, 1634
rti::config::LogCategory_def, 1682
all_categories, 1683
api, 1683
communication, 1683
database, 1683
discovery, 1683
entities, 1683
platform, 1682
security, 1683
type, 1682
user, 1683
rti::config::Logger, 1683
alert, 1689, 1691
critical, 1689, 1691
debug, 1690, 1692
emergency, 1689, 1690
error, 1689, 1691
informational, 1690, 1692
instance, 1685
notice, 1690, 1691
output_file, 1686, 1687
output_file_set, 1687
output_handler, 1687
print_format, 1688
print_format_by_log_level, 1688, 1689
reset_output_handler, 1688
verbosity, 1685, 1686
verbosity_by_category, 1685, 1686
warning, 1690, 1691
rti::config::LogLevel_def, 1692
EXCEPTION, 1693
FATAL_ERROR, 1693
STATUS_ALL, 1693
STATUS_LOCAL, 1693
STATUS_REMOTE, 1693
type, 1693
WARNING, 1693
rti::config::LogMessage, 1693
facility, 1695
is_license_message, 1695
is_security_message, 1694
level, 1694
message_id, 1695
text, 1694
timestamp, 1695
rti::config::PrintFormat_def, 1960
DEBUG, 1961
DEFAULT, 1961
MAXIMAL, 1961
MINIMAL, 1961
TIMESTAMPED, 1961
type, 1960
VERBOSE, 1961
VERBOSE_TIMESTAMPED, 1961
rti::config::ScopedLoggerVerbosity, 2319
~ScopedLoggerVerbosity, 2319
ScopedLoggerVerbosity, 2319
rti::config::Verbosity_def, 2613
exception, 2614
silent, 2614
status_all, 2614
status_local, 2614
status_remote, 2614
type, 2613
warning, 2614
rti::core, 701
bind_and_manage_listener, 709
bind_listener, 708
default_qos_provider_params, 710
fill_array, 707
initialize_native_array, 707
LocatorKind, 706
MonitoringMetricSelectionSeq, 706
operator<<, 708, 710
ThreadSettingsCpuRotationKind, 706
TransportClassId, 706
TransportInfoSeq, 707
rti::core::AllocationSettings, 803
AllocationSettings, 804
AUTO_COUNT, 805
incremental_count, 805
initial_count, 804
max_count, 804, 805
rti::core::bounded_sequence< T, MaxLength >, 887
allocator_type, 891
at, 897
back, 898
begin, 899
bounded_sequence, 893–895
capacity, 902
cbegin, 899
cend, 900
clear, 902

const_iterator, 892
const_pointer, 892
const_reference, 892
const_reverse_iterator, 893
crbegin, 900
crend, 901
data, 898, 899
difference_type, 891
empty, 901
end, 899, 900
erase, 903
front, 898
insert, 903
iterator, 892
max_size, 901
operator=, 895, 896
operator[], 897
pointer, 892
pop_back, 904
push_back, 903, 904
rbegin, 900
reference, 891
rend, 901
reserve, 902
resize, 904, 905
reverse_iterator, 893
shrink_to_fit, 902
size, 901
size_type, 891
swap, 905
value_type, 891
vector_type, 891
rti::core::builtin_profiles, 711
rti::core::builtin_profiles::qos_lib, 712
rti::core::builtin_profiles::qos_lib_exp, 714
rti::core::builtin_profiles::qos_snippet_lib, 716
rti::core::ChannelSettings, 921
ChannelSettings, 922
filter_expression, 923
multicast_settings, 922, 923
priority, 924
rti::core::CoherentSetInfo, 934
coherent_set_sequence_number, 936, 937
CoherentSetInfo, 935
group_coherent_set_sequence_number, 937, 938
group_guid, 936
incomplete_coherent_set, 938
operator<<, 939
unknown, 938
rti::core::CompressionIdMask, 940
all, 941
bzip2, 942
CompressionIdMask, 941
default_publication, 942
default_subscription, 942
lz4, 943
MaskType, 941
none, 942
zlib, 942
rti::core::CompressionSettings, 943
compression_ids, 945
compression_level_best_compression, 945
compression_level_best_speed, 945
compression_level_default, 945
CompressionSettings, 944
writer_compression_level, 946
writer_compression_threshold, 947
rti::core::cond::AsyncWaitSet, 840
AsyncWaitSet, 844, 845
attach_condition, 848
conditions, 853
detach_condition, 849, 850
operator+=, 851
operator-=, 851
property, 852
start, 845
stop, 846, 847
unlock_condition, 852
rti::core::cond::AsyncWaitSetCompletionToken, 853
AsyncWaitSetCompletionToken, 855
wait, 855
rti::core::cond::AsyncWaitSetListener, 856
on_thread_deleted, 857
on_thread_spawned, 857
on_wait_timeout, 857
rti::core::cond::AsyncWaitSetProperty, 858
AsyncWaitSetProperty, 859
level, 860
thread_name_prefix, 861
thread_pool_size, 862
thread_settings, 861
wait_timeout, 860
waitset_property, 859
rti::core::cond::NoOpAsyncWaitSetListener, 1836
on_thread_deleted, 1837
on_thread_spawned, 1837
on_wait_timeout, 1837
rti::core::cond::WaitSetProperty, 2627
max_event_count, 2629
max_event_delay, 2629, 2630
WaitSetProperty, 2628, 2629
rti::core::ContentFilterProperty, 960
content_filter_topic_name, 961
expression_parameters, 961
filter_class_name, 961
filter_expression, 961
related_topic_name, 961
rti::core::Cookie, 965

Cookie, 966
CookieSeq, 967
operator<<, 967
to_pointer, 966
value, 966
rti::core::DataReaderResourceLimitsInstanceReplacementSettings, 1108
alive_instance_removal, 1110
DataReaderResourceLimitsInstanceReplacementSettings, 1109
disposed_instance_removal, 1110
no_writers_instance_removal, 1110, 1111
rti::core::DataWriterShmemRefTransferModeSettings, 1251
DataWriterShmemRefTransferModeSettings, 1252
enable_data_consistency_check, 1252, 1253
rti::core::EndpointGroup, 1515
EndpointGroup, 1515
quorum_count, 1516
role_name, 1515, 1516
rti::core::Guid, 1596
automatic, 1597
Guid, 1597
LENGTH, 1599
operator<, 1598
operator<<, 1599
operator<=, 1598
operator>, 1598
operator>=, 1598
operator[], 1597, 1598
unknown, 1597
zero, 1597
rti::core::ListenerBinder< Entity, Listener >, 1641
bind_and_manage_listener, 1644, 1645
bind_listener, 1644
entity, 1643
get, 1643
listener, 1643
rti::core::Locator, 1673
address, 1675
Invalid, 1675
kind, 1674
Locator, 1674
LocatorSeq, 1675
port, 1674, 1675
rti::core::LocatorFilterElement, 1678
filter_expression, 1680
LocatorFilterElement, 1679
locators, 1679, 1680
rti::core::LocatorKind_def, 1681
INVALID, 1681
SHMEM, 1681
type, 1681
UDPv4, 1681
UDPv4_WAN, 1681
UDPv6, 1681
rti::core::LongDouble, 1696
LongDouble, 1696
operator[], 1697
rti::core::MonitoringDedicatedParticipantSettings, 1712
collector_initial_peers, 1715, 1716
domain_id, 1713, 1714
enable, 1713
MonitoringDedicatedParticipantSettings, 1712
participant_qos_profile_name, 1714, 1715
rti::core::MonitoringDistributionSettings, 1716
dedicated_participant, 1719
event_settings, 1720
logging_settings, 1721, 1722
MonitoringDistributionSettings, 1718
periodic_settings, 1720, 1721
publisher_qos_profile_name, 1718, 1719
rti::core::MonitoringEventDistributionSettings, 1722
concurrency_level, 1723, 1724
datawriter_qos_profile_name, 1724, 1725
MonitoringEventDistributionSettings, 1723
publication_period, 1726, 1727
thread, 1726
rti::core::MonitoringLoggingDistributionSettings, 1727
concurrency_level, 1729
datawriter_qos_profile_name, 1730, 1731
max_historical_logs, 1729, 1730
MonitoringLoggingDistributionSettings, 1728
publication_period, 1732
thread, 1731, 1732
rti::core::MonitoringLoggingForwardingSettings, 1733
middleware_forwarding_level, 1734
security_event_forwarding_level, 1734, 1735
service_forwarding_level, 1735
user_forwarding_level, 1736
rti::core::MonitoringMetricSelection, 1736
disabled_metrics_selection, 1739, 1740
enabled_metrics_selection, 1738, 1739
MonitoringMetricSelection, 1737
resource_selection, 1737, 1738
rti::core::MonitoringPeriodicDistributionSettings, 1740
datawriter_qos_profile_name, 1742, 1743
MonitoringPeriodicDistributionSettings, 1741
polling_period, 1744
thread, 1743
rti::core::MonitoringTelemetryData, 1744
logs, 1746
metrics, 1745, 1746
rti::core::optional_value< T >, 1892
get, 1896
has_value, 1894
is_set, 1894
operator bool, 1894

operator*, 1896, 1897
operator->, 1897
optional_value, 1893, 1894
reset, 1895
swap, 1898
value, 1895
rti::core::PersistentStorageSettings, 1927
enable, 1929
file_name, 1929–1931
journal_kind, 1932
PersistentStorageSettings, 1929
reader_checkpoint_frequency, 1937, 1938
restore, 1934
synchronization_kind, 1933
trace_file_name, 1931, 1932
vacuum, 1933
writer_instance_cache_allocation, 1934, 1935
writer_memory_state, 1936, 1937
writer_sample_cache_allocation, 1935, 1936
rti::core::pointer< T >, 1938
get, 1939
operator!=, 1940
operator=, 1940
operator==, 1940
pointer, 1939
set, 1939
rti::core::policy::AcknowledgmentKind_def, 797
APPLICATION_AUTO, 798
APPLICATION_EXPLICIT, 798
APPLICATION_ORDERED, 798
PROTOCOL, 797
type, 797
rti::core::policy::AsynchronousPublisher, 832
asynchronous_batch_thread, 837, 838
AsynchronousPublisher, 835
disable_asynchronous_batch, 836, 837
disable_asynchronous_write, 835
disable_topic_query_publication, 838
Disabled, 840
Enabled, 839
thread, 835, 836
topic_query_publication_thread, 838, 839
rti::core::policy::Availability, 868
Availability, 870
enable_required_subscriptions, 871
max_data_availability_waiting_time, 871, 872
max_endpoint_availability_waiting_time, 872, 873
required_matched_endpoint_groups, 873, 874
rti::core::policy::Batch, 880
Batch, 881
Disabled, 881
enable, 882
Enabled, 881
EnabledWithMaxDataBytes, 882
EnabledWithMaxSamples, 882
max_data_bytes, 882, 883
max_flush_delay, 884, 885
max_samples, 883, 884
source_timestamp_resolution, 885, 886
thread_safe_write, 886
rti::core::policy::BuiltinTopicReaderResourceLimits, 906
BuiltinTopicReaderResourceLimits, 908
disable_fragmentation_support, 913
dynamically_allocate_fragmented_samples, 916, 917
initial_fragmented_samples, 914, 915
initial_infos, 910
initial_outstanding_reads, 911
initial_samples, 909
max_fragmented_samples, 913, 914
max_fragmented_samples_per_remote_writer, 915
max_fragments_per_sample, 916
max_infos, 911
max_outstanding_reads, 912
max_samples, 909, 910
max_samples_per_read, 912
rti::core::policy::CdrPaddingKind_def, 920
AUTO, 921
NOT_SET, 921
type, 921
ZERO, 921
rti::core::policy::Database, 969
cleanup_period, 972, 973
initial_records, 974
initial_weak_references, 975, 976
max_skiplist_level, 974
max_weak_references, 974, 975
shutdown_cleanup_period, 973
shutdown_timeout, 972
thread, 971, 972
rti::core::policy::DataReaderInstanceRemovalKind_def, 1053
ANY_INSTANCE, 1055
EMPTY_INSTANCES, 1054
FULLY_PROCESSED_INSTANCES, 1055
NO_INSTANCE, 1054
type, 1054
rti::core::policy::DataReaderProtocol, 1059
DataReaderProtocol, 1060
disable_positive_acks, 1062, 1063
expects_inline_qos, 1062
propagate_dispose_of_unregistered_instances, 1063
propagate_unregister_of_disposed_instances, 1064
rtps_object_id, 1061
rtps_reliable_reader, 1064, 1065
virtual_guid, 1060, 1061
rti::core::policy::DataReaderResourceLimits, 1081

autopurge_remote_not_alive_writer_delay, 1103, 1104
autopurge_remote_virtual_writer_delay, 1104, 1105
DataReaderResourceLimits, 1085
disable_fragmentation_support, 1092
dynamically_allocate_fragmented_samples, 1095
initial_fragmented_samples, 1093
initial_infos, 1089, 1090
initial_outstanding_reads, 1090
initial_remote_virtual_writers, 1097, 1098
initial_remote_virtual_writers_per_instance, 1099
initial_remote_writers, 1088
initial_remote_writers_per_instance, 1089
initial_topic_queries, 1102
instance_replacement, 1106, 1107
keep_minimum_state_for_instances, 1101, 1102
lazily_allocate_batch_resources, 1105
max_app_ack_response_length, 1101
max_fragmented_samples, 1092, 1093
max_fragmented_samples_per_remote_writer, 1094
max_fragments_per_sample, 1094, 1095
max_infos, 1087, 1088
max_outstanding_reads, 1091
max_query_condition_filters, 1100
max_remote_virtual_writers, 1097
max_remote_virtual_writers_per_instance, 1098
max_remote_writers, 1085, 1086
max_remote_writers_per_instance, 1086
max_remote_writers_per_sample, 1099, 1100
max_samples_per_read, 1091
max_samples_per_remote_writer, 1087
max_topic_queries, 1103
max_total_instances, 1096
shmem_ref_transfer_mode_attached_segment_allocation, 1106
rti::core::policy::DataWriterProtocol, 1210
DataWriterProtocol, 1211
disable_inline_keyhash, 1214, 1215
disable_positive_acks, 1214
initial_virtual_sequence_number, 1218
propagate_app_ack_with_no_response, 1216, 1217
push_on_write, 1213
rtps_object_id, 1212, 1213
rtps_reliable_writer, 1217, 1218
serialize_key_with_dispose, 1215, 1216
virtual_guid, 1212
rti::core::policy::DataWriterResourceLimits, 1235
autoregister_instances, 1242, 1243
cookie_max_length, 1241
DataWriterResourceLimits, 1237
initial_active_topic_queries, 1246
initial_batches, 1239, 1240
initial_concurrent_blocking_threads, 1238
initial_remote_readers, 1244, 1245
initial_virtual_writers, 1243
initialize_writer_loaned_sample, 1248, 1249
instance_replacement, 1241
max_active_topic_queries, 1246, 1247
max_app_ack_remote_readers, 1245
max_batches, 1240
max_concurrent_blocking_threads, 1238
max_remote_reader_filters, 1239
max_remote_readers, 1244, 1245
max_virtual_writers, 1244
replace_empty_instances, 1242
writer_loaned_sample_allocation, 1247, 1248
rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_def, 1249
ALIVE, 1251
ALIVE_OR_DISPOSED, 1251
ALIVE_THEN_DISPOSED, 1251
DISPOSED, 1251
DISPOSED_THEN_ALIVE, 1251
type, 1250
UNREGISTERED, 1251
rti::core::policy::DataWriterTransferMode, 1253
DataWriterTransferMode, 1254
shmem_ref_settings, 1254, 1255
ShmemRefSettings, 1254
rti::core::policy::DestinationOrderScopeKind_def, 1264
INSTANCE, 1265
TOPIC, 1265
type, 1265
rti::core::policy::Discovery, 1265
accept_unknown_peers, 1270
Discovery, 1267
enable_endpoint_discovery, 1271
enabled_transports, 1267
initial_peers, 1268
metatraffic_transport_priority, 1269, 1270
multicast_receive_addresses, 1268, 1269
rti::core::policy::DiscoveryConfig, 1271
asynchronous_publisher, 1299, 1300
builtin_discovery_plugins, 1293, 1294
default_domain_announcement_period, 1300
DiscoveryConfig, 1279
dns_tracker_polling_period, 1311, 1312
enabled_builtin_channels, 1294
endpoint_type_object_lb_serialization_threshold, 1310, 1311
ignore_default_domain_announcements, 1301
initial_participant_announcements, 1282
locator_reachability_assert_period, 1305
locator_reachability_change_detection_period, 1306, 1307
locator_reachability_lease_duration, 1306
max_initial_participant_announcement_period, 1284, 1285

max_liveliness_loss_detection_period, 1281, 1282
min_initial_participant_announcement_period, 1283, 1284
new_remote_participant_announcements, 1283
participant_announcement_period, 1280
participant_configuration_reader, 1312, 1313
participant_configuration_reader_resource_limits, 1313, 1314
participant_configuration_writer, 1314, 1315
participant_configuration_writer_data_lifecycle, 1316
participant_configuration_writer_publish_mode, 1316, 1317
participant_liveliness_assert_period, 1279, 1280
participant_liveliness_lease_duration, 1279
participant_message_reader, 1295, 1296
participant_message_reader_reliability_kind, 1295
participant_message_writer, 1296, 1297
participant_reader_resource_limits, 1285, 1286
publication_reader, 1286, 1287
publication_reader_resource_limits, 1287
publication_writer, 1289, 1290
publication_writer_data_lifecycle, 1291
publication_writer_publish_mode, 1298
remote_participant_purge_kind, 1281
request_types_filter, 1317, 1318
secure_volatile_reader, 1309, 1310
secure_volatile_writer, 1307, 1308
secure_volatile_writer_publish_mode, 1309
service_request_reader, 1304, 1305
service_request_writer, 1301, 1302
service_request_writer_data_lifecycle, 1303
service_request_writer_publish_mode, 1303, 1304
subscription_reader, 1288
subscription_reader_resource_limits, 1289
subscription_writer, 1291, 1292
subscription_writer_data_lifecycle, 1293
subscription_writer_publish_mode, 1298, 1299
rti::core::policy::DiscoveryConfigBuiltinChannelKindMask, 1319
DiscoveryConfigBuiltinChannelKindMask, 1320
MaskType, 1320
service_request, 1321
type_lookup_service, 1321
type_lookup_service_reply, 1321
type_lookup_service_request, 1321
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask, 1322
DiscoveryConfigBuiltinPluginKindMask, 1323
DPSE, 1325
MaskType, 1323
none, 1324
SDP, 1325
SDP2, 1325
SEDP, 1324
SPDP, 1324
SPDP2, 1324
rti::core::policy::DomainParticipantResourceLimits, 1391
channel_filter_expression_max_length, 1424
channel_seq_max_length, 1423
content_filter_allocation, 1404
content_filter_hash_buckets, 1412
content_filtered_topic_allocation, 1403
content_filtered_topic_hash_buckets, 1412
contentfilter_property_max_length, 1422, 1423
deserialized_type_object_dynamic_allocation_threshold, 1420, 1421
flow_controller_allocation, 1406
flow_controller_hash_buckets, 1413
ignored_entity_allocation, 1403
ignored_entity_hash_buckets, 1411
ignored_entity_replacement_kind, 1429
local_publisher_allocation, 1399
local_publisher_hash_buckets, 1407
local_reader_allocation, 1398
local_reader_hash_buckets, 1407
local_subscriber_allocation, 1399
local_subscriber_hash_buckets, 1408
local_topic_allocation, 1399, 1400
local_topic_hash_buckets, 1408
local_writer_allocation, 1398
local_writer_hash_buckets, 1406
matching_reader_writer_pair_allocation, 1402
matching_reader_writer_pair_hash_buckets, 1411
matching_writer_reader_pair_allocation, 1402
matching_writer_reader_pair_hash_buckets, 1410
max_endpoint_group_cumulative_characters, 1428
max_endpoint_groups, 1427
max_gather_destinations, 1413
max_partition_cumulative_characters, 1417
max_partitions, 1417
outstanding_asynchronous_sample_allocation, 1405
participant_property_list_max_length, 1424
participant_property_string_max_length, 1425
participant_user_data_max_length, 1414
publisher_group_data_max_length, 1415
query_condition_allocation, 1405
read_condition_allocation, 1404
reader_data_tag_list_max_length, 1432
reader_data_tag_string_max_length, 1432
reader_property_list_max_length, 1426
reader_property_string_max_length, 1427
reader_user_data_max_length, 1416
remote_participant_allocation, 1401
remote_participant_hash_buckets, 1410
remote_reader_allocation, 1401
remote_reader_hash_buckets, 1409
remote_topic_query_allocation, 1429, 1430
remote_topic_query_hash_buckets, 1430

remote_writer_allocation, 1400
remote_writer_hash_buckets, 1409
serialized_type_object_dynamic_allocation_threshold, 1421, 1422
shmem_ref_transfer_mode_max_segments, 1433
subscriber_group_data_max_length, 1415
topic_data_max_length, 1414
transport_info_list_max_length, 1428
type_code_max_serialized_length, 1418
type_object_max_deserialized_length, 1420
type_object_max_serialized_length, 1419
writer_data_tag_list_max_length, 1431
writer_data_tag_string_max_length, 1431
writer_property_list_max_length, 1425
writer_property_string_max_length, 1426
writer_user_data_max_length, 1416
rti::core::policy::EntityName, 1530
EntityName, 1531
name, 1531, 1532
role_name, 1532
rti::core::policy::Event, 1541
Event, 1542
initial_count, 1543
max_count, 1543, 1544
thread, 1542, 1543
rti::core::policy::IgnoredEntityReplacementKind_def, 1607
type, 1607
rti::core::policy::LocatorFilter, 1676
Filter, 1677
filter_name, 1678
FilterSeq, 1677
locator_filters, 1677, 1678
LocatorFilter, 1677
rti::core::policy::Monitoring, 1706
application_name, 1709, 1710
Disabled, 1707
distribution_settings, 1710, 1711
enable, 1708
Enabled, 1707
Monitoring, 1707
telemetry_data, 1711
rti::core::policy::MulticastMapping, 1747
addresses, 1748
mapping_function, 1749
topic_expression, 1748, 1749
rti::core::policy::MultiChannel, 1749
channels, 1751, 1752
filter_name, 1752
MultiChannel, 1751
rti::core::policy::Property, 1969
Entry, 1972
exists, 1973
get, 1973
get_all, 1975
propagate, 1975
Property, 1972
remove, 1974
set, 1973–1975
size, 1974
try_get, 1973
rti::core::policy::PublishMode, 2013
Asynchronous, 2015, 2016
flow_controller_name, 2016
kind, 2016
priority, 2017, 2018
PublishMode, 2015
Synchronous, 2015
rti::core::policy::PublishModeKind_def, 2018
ASYNCHRONOUS, 2019
SYNCHRONOUS, 2019
type, 2019
rti::core::policy::ReceiverPool, 2144
buffer_alignment, 2148
buffer_size, 2147
ReceiverPool, 2145
thread, 2146, 2147
rti::core::policy::RemoteParticipantPurgeKind_def, 2164
LIVELINESS_BASED, 2165
NO_PURGE, 2165
type, 2164
rti::core::policy::RtpsReliableWriterProtocol, 2229
disable_positive_acks_decrease_sample_keep_duration_factor, 2248
disable_positive_acks_enable_adaptive_sample_keep_duration, 2247
disable_positive_acks_increase_sample_keep_duration_factor, 2248, 2249
disable_positive_acks_max_sample_keep_duration, 2246
disable_positive_acks_min_sample_keep_duration, 2245, 2246
disable_repair_piggyback_heartbeat, 2255
enable_multicast_periodic_heartbeat, 2254
fast_heartbeat_period, 2235, 2236
heartbeat_period, 2234, 2235
heartbeats_per_max_samples, 2240, 2241
high_watermark, 2233, 2234
inactivate_nonprogressing_readers, 2239, 2240
late_joiner_heartbeat_period, 2236, 2237
low_watermark, 2232, 2233
max_bytes_per_nack_response, 2244, 2245
max_heartbeat_retries, 2239
max_nack_response_delay, 2242, 2243
max_send_window_size, 2250, 2251
min_nack_response_delay, 2242
min_send_window_size, 2249, 2250
multicast_resend_threshold, 2254
nack_suppression_duration, 2243, 2244

RtpsReliableWriterProtocol, 2232
samples_per_virtual_heartbeat, 2238
send_window_decrease_factor, 2253
send_window_increase_factor, 2252
send_window_update_period, 2251, 2252
virtual_heartbeat_period, 2237
rti::core::policy::RtpsReservedPortKindMask, 2255
builtin_multicast, 2258
builtin_unicast, 2257
MaskType, 2256
RtpsReservedPortKindMask, 2257
user_multicast, 2258
user_unicast, 2258
rti::core::policy::Service, 2350
AdminConsoleService, 2353
DatabaseIntegrationService, 2353
kind, 2354
NoService, 2352
ObservabilityCollectorService, 2353
PersistenceService, 2352
QueuingService, 2352
RecordingService, 2353
ReplayService, 2353
RoutingService, 2352
Service, 2351, 2352
WebIntegrationService, 2353
rti::core::policy::ServiceKind_def, 2357
ADMIN_CONSOLE, 2358
DATABASE_INTEGRATION, 2358
NO_SERVICE, 2358
OBSERVABILITY_COLLECTOR, 2358
PERSISTENCE, 2358
QUEUING, 2358
RECORDING, 2358
REPLAY, 2358
ROUTING, 2358
type, 2358
WEB_INTEGRATION, 2358
rti::core::policy::SystemResourceLimits, 2443
initial_objects_per_thread, 2445, 2446
max_objects_per_thread, 2445, 2446
SystemResourceLimits, 2444
rti::core::policy::TopicQueryDispatch, 2522
enable, 2524
publication_period, 2525
samples_per_period, 2525
TopicQueryDispatch, 2524
rti::core::policy::TransportBuiltin, 2533
All, 2535
mask, 2536
None, 2535
Shmem, 2535
SHMEM_ALIAS, 2536
TransportBuiltin, 2535
UDPv4, 2535
UDPv4_ALIAS, 2537
UDPv4_WAN, 2536
UDPv4_WAN_ALIAS, 2537
UDPv6, 2536
UDPv6_ALIAS, 2537
rti::core::policy::TransportBuiltinMask, 2537
all, 2539
MaskType, 2538
none, 2539
shmem, 2539
TransportBuiltinMask, 2538, 2539
udpv4, 2539
udpv4_wan, 2540
udpv6, 2540
rti::core::policy::TransportMulticast, 2543
kind, 2546
settings, 2545
TransportMulticast, 2544, 2545
rti::core::policy::TransportMulticastKind_def, 2547
AUTOMATIC, 2547
type, 2547
UNICAST, 2547
rti::core::policy::TransportMulticastMapping, 2547
mappings, 2549
rti::core::policy::TransportSelection, 2554
enabled_transports, 2555, 2556
TransportSelection, 2555
rti::core::policy::TransportUnicast, 2556
settings, 2558, 2559
TransportUnicast, 2558
rti::core::policy::TypeSupport, 2574
cdr_padding_kind, 2576
plugin_data, 2575
TypeSupport, 2575
rti::core::policy::WireProtocol, 2630
check_crc, 2640
compute_crc, 2640
participant_id, 2635
rtps_app_id, 2636, 2637
RTPS_AUTO_ID, 2641
rtps_auto_id_kind, 2639
rtps_host_id, 2636
rtps_instance_id, 2637, 2638
rtps_reserved_port_mask, 2639
rtps_well_known_ports, 2638
WireProtocol, 2634
rti::core::policy::WireProtocolAutoKind_def, 2641
RTPS_AUTO_ID_FROM_IP, 2642
RTPS_AUTO_ID_FROM_MAC, 2642
RTPS_AUTO_ID_FROM_UUID, 2642
type, 2642
rti::core::ProductVersion, 1967
current, 1969

major_version, 1968
minor_version, 1968
ProductVersion, 1967
release_version, 1968
revision_version, 1968
to_string, 1968
unknown, 1969
rti::core::ProtocolVersion, 1975
current, 1977
major_version, 1976
minor_version, 1977
ProtocolVersion, 1976
rti::core::qos_print_all_t, 2019
rti::core::QosPrintFormat, 2021
indent, 2022
is_standalone, 2024
print_private, 2023
QosPrintFormat, 2022
rti::core::QosProviderParams, 2049
ignore_environment_profile, 2052
ignore_resource_profile, 2052, 2053
ignore_user_profile, 2051
QosProviderParams, 2050
string_profile, 2050
url_profile, 2050, 2051
rti::core::Result< T >, 2209
get, 2211, 2212
get_if_ok, 2212, 2213
get_return_code, 2212
is_ok, 2212
Result, 2210, 2211
throw_if_error, 2213
rti::core::RtpsReliableReaderProtocol, 2221
app_ack_period, 2227
heartbeat_suppression_duration, 2224, 2225
max_heartbeat_response_delay, 2224
min_app_ack_response_keep_duration, 2227, 2228
min_heartbeat_response_delay, 2223
nack_period, 2225
receive_window_size, 2226
round_trip_time, 2226
RtpsReliableReaderProtocol, 2223
samples_per_app_ack, 2228, 2229
rti::core::RtpsWellKnownPorts, 2258
builtin_multicast_port_offset, 2263, 2264
builtin_unicast_port_offset, 2264
domain_id_gain, 2262
participant_id_gain, 2263
port_base, 2262
RtpsWellKnownPorts, 2261
user_multicast_port_offset, 2264
user_unicast_port_offset, 2265
rti::core::SampleFlag, 2278
discovery_service_sample, 2281
incomplete_snapshot_topic_query, 2281
intermediate_reply_sequence, 2280
intermediate_topic_query_sample, 2281
last_shared_reader_queue, 2280
MaskType, 2279
redelivered, 2280
replicate, 2280
SampleFlag, 2279, 2280
writer_removed_batch_sample, 2281
rti::core::SampleIdentity, 2282
automatic, 2284
operator<<, 2284
SampleIdentity, 2283
sequence_number, 2284
unknown, 2284
writer_guid, 2283
rti::core::SequenceNumber, 2335
automatic, 2338
high, 2339
low, 2339
maximum, 2338
operator<, 2341
operator<<, 2342
operator<=, 2342
operator>, 2342
operator>=, 2342
operator+, 2340
operator++, 2341
operator+=, 2340
operator-, 2340
operator--, 2341
operator-=, 2340
SequenceNumber, 2337, 2338
unknown, 2338
value, 2339, 2340
zero, 2338
rti::core::ServiceRequestId_def, 2361
INSTANCE_STATE, 2362
LOCATOR_REACHABILITY, 2362
MONITORING_LIBRARY_COMMAND, 2362
MONITORING_LIBRARY_REPLY, 2362
TOPIC_QUERY, 2362
type, 2362
UNKNOWN, 2362
rti::core::status::DataReaderCacheStatus, 1046
alive_instance_count, 1051
alive_instance_count_peak, 1051
compressed_sample_count, 1053
content_filter_dropped_sample_count, 1049
detached_instance_count, 1052
detached_instance_count_peak, 1053
disposed_instance_count, 1052
disposed_instance_count_peak, 1052
expired_dropped_sample_count, 1049

no_writers_instance_count, 1051
no_writers_instance_count_peak, 1052
old_source_timestamp_dropped_sample_count, 1048
ownership_dropped_sample_count, 1049
replaced_dropped_sample_count, 1050
sample_count, 1048
sample_count_peak, 1048
time_based_filter_dropped_sample_count, 1049
tolerance_source_timestamp_dropped_sample_count, 1048
total_samples_dropped_by_instance_replacement, 1051
virtual_duplicate_dropped_sample_count, 1050
writer_removed_batch_sample_dropped_sample_count, 1050
rti::core::status::DataReaderProtocolStatus, 1065
dropped_fragment_count, 1071
duplicate_sample_bytes, 1068
duplicate_sample_count, 1067
filtered_sample_bytes, 1068
filtered_sample_count, 1068
first_available_sample_sequence_number, 1070
last_available_sample_sequence_number, 1070
last_committed_sample_sequence_number, 1070
out_of_range_rejected_sample_count, 1072
reassembled_sample_count, 1071
received_fragment_count, 1071
received_gap_bytes, 1069
received_gap_count, 1069
received_heartbeat_bytes, 1068
received_heartbeat_count, 1068
received_sample_bytes, 1067
received_sample_count, 1066
rejected_sample_count, 1070
sent_ack_bytes, 1069
sent_ack_count, 1069
sent_nack_bytes, 1069
sent_nack_count, 1069
sent_nack_fragment_bytes, 1072
sent_nack_fragment_count, 1071
uncommitted_sample_count, 1071
rti::core::status::DataWriterCacheStatus, 1199
alive_instance_count, 1200
alive_instance_count_peak, 1201
disposed_instance_count, 1201
disposed_instance_count_peak, 1201
sample_count, 1200
sample_count_peak, 1200
unregistered_instance_count, 1201
unregistered_instance_count_peak, 1201
rti::core::status::DataWriterProtocolStatus, 1219
filtered_sample_bytes, 1221
filtered_sample_count, 1221
first_available_sample_sequence_number, 1223
first_available_sample_virtual_sequence_number, 1224
first_unacknowledged_sample_sequence_number, 1224
first_unacknowledged_sample_subscription_handle, 1225
first_unacknowledged_sample_virtual_sequence_number, 1224
first_unelapsed_keep_duration_sample_sequence_number, 1225
last_available_sample_sequence_number, 1224
last_available_sample_virtual_sequence_number, 1224
pulled_fragment_bytes, 1226
pulled_fragment_count, 1225
pulled_sample_bytes, 1222
pulled_sample_count, 1221
pushed_fragment_bytes, 1225
pushed_fragment_count, 1225
pushed_sample_bytes, 1220
pushed_sample_count, 1220
received_ack_bytes, 1222
received_ack_count, 1222
received_nack_bytes, 1222
received_nack_count, 1222
received_nack_fragment_bytes, 1226
received_nack_fragment_count, 1226
rejected_sample_count, 1223
send_window_size, 1223
sent_gap_bytes, 1223
sent_gap_count, 1223
sent_heartbeat_bytes, 1221
sent_heartbeat_count, 1221
rti::core::status::DomainParticipantProtocolStatus, 1383
corrupted_rtps_message_count, 1383
corrupted_rtps_message_count_change, 1383
last_corrupted_message_timestamp, 1384
rti::core::status::EventCount< IntegerType >, 1544
change, 1545
operator<<, 1545
total, 1545
rti::core::status::InvalidLocalAdvanceNoticeStatus, 1621
expiration_code, 1621
expiration_message, 1621
expiration_time, 1621
rti::core::status::ReliableReaderActivityChangedStatus, 2157
active_count, 2158
inactive_count, 2158
last_instance_handle, 2159
rti::core::status::ReliableWriterCacheChangedStatus, 2159
empty_reliable_writer_cache, 2160

full_reliable_writer_cache, 2160
high_watermark_reliable_writer_cache, 2161
low_watermark_reliable_writer_cache, 2160
replaced_unacknowledged_sample_count, 2161
unacknowledged_sample_count, 2161
unacknowledged_sample_count_peak, 2161
rti::core::status::SampleLostState, 2296
lost_by_availability_waiting_time, 2301
lost_by_decode_failure, 2302
lost_by_deserialization_failure, 2302
lost_by_incomplete_coherent_set, 2299
lost_by_instances_limit, 2299
lost_by_large_coherent_set, 2300
lost_by_out_of_memory, 2302
lost_by_remote_writers_per_instance_limit, 2299
lost_by_remote_writers_per_sample_limit, 2301
lost_by_remote_writers_per_virtual_queue_limit, 2301
lost_by_samples_limit, 2303
lost_by_samples_per_instance_limit, 2303
lost_by_samples_per_remote_writer_limit, 2300
lost_by_unknown_instance, 2302
lost_by_virtual_writers_limit, 2300
lost_by_writer, 2299
not_lost, 2299
SampleLostState, 2298
rti::core::status::ServiceRequestAcceptedStatus, 2360
current_count, 2360
last_request_handle, 2361
service_id, 2361
total_count, 2360
rti::core::ThreadSettings, 2449
cpu_list, 2451
cpu_rotation, 2452
mask, 2450
priority, 2450, 2451
stack_size, 2451
ThreadSettings, 2450
rti::core::ThreadSettingsCpuRotationKind_def, 2453
NO_ROTATION, 2454
ROUND_ROBIN, 2454
type, 2453
rti::core::ThreadSettingsKindMask, 2454
cancel_asynchronous, 2456
floating_point, 2456
MaskType, 2455
priority_enforce, 2456
realtime_priority, 2456
stdio, 2456
ThreadSettingsKindMask, 2455
rti::core::TransportClassId_def, 2540
INVALID, 2541
RESERVED_RANGE, 2541
SHMEM, 2541
SHMEM_510, 2541
TCPV4_LAN, 2541
TCPV4_WAN, 2541
TLSV4_LAN, 2541
TLSV4_WAN, 2541
type, 2541
UDPv4, 2541
UDPv4_WAN, 2541
UDPv6, 2541
UDPv6_510, 2541
rti::core::TransportInfo, 2541
class_id, 2542, 2543
message_size_max, 2543
TransportInfo, 2542
rti::core::TransportMulticastSettings, 2549
receive_address, 2551
receive_port, 2551
TransportMulticastSettings, 2550
transports, 2550
rti::core::TransportUnicastSettings, 2559
receive_port, 2561
transports, 2561
TransportUnicastSettings, 2560
rti::core::UnregisterThreadOnExit, 2589
~UnregisterThreadOnExit, 2590
rti::core::VendorId, 2612
unknown, 2612
value, 2612
VendorId, 2612
rti::core::xtypes, 718
can_convert, 723
convert, 720, 722, 723
create_type_from_tuple, 725
DynamicTypePrintKind, 720
from_cdr_buffer, 721
get_tuple, 724
idl, 720
json_schema, 720
print_idl, 725
resolve_alias, 721
set_tuple, 724
to_cdr_buffer, 721
to_string, 726
xml, 720
rti::core::xtypes::DynamicDataInfo, 1494
member_count, 1494
stored_size, 1495
rti::core::xtypes::DynamicDataMemberInfo, 1495
element_count, 1496
element_kind, 1497
member_exists, 1497
member_index, 1496
member_kind, 1496
member_name, 1496

rti::core::xtypes::DynamicDataProperty, 1497
buffer_initial_size, 1498
buffer_max_size, 1499
rti::core::xtypes::DynamicDataTypeSerializationProperty, 1500
DEFAULT, 1503
DynamicDataTypeSerializationProperty, 1501
max_size_serialized, 1501
min_size_serialized, 1501, 1502
skip_deserialization, 1502, 1503
trim_to_size, 1502
rti::core::xtypes::DynamicTypePrintFormatProperty, 1509
DynamicTypePrintFormatProperty, 1510
indent, 1510
print_complete_type, 1513
print_kind, 1512, 1513
print_ordinals, 1511
rti::core::xtypes::LoanedDynamicData, 1656
~LoanedDynamicData, 1657
get, 1658
LoanedDynamicData, 1658
operator const DynamicData &, 1659
operator DynamicData &, 1658
operator=, 1659
return_loan, 1658
rti::core::xtypes::octet_tag_t, 1871
rti::domain, 727
banish_ignored_participants, 728
discovered_participant_subject_name, 728
discovered_participants_from_subject_name, 730
find_participant_by_name, 731
find_participants, 731, 732
find_type, 732
register_type, 733, 734
rti::domain::DomainParticipantConfigParams, 1371
domain_entity_qos_library_name, 1375, 1376
domain_entity_qos_profile_name, 1376
domain_id, 1373
DOMAIN_ID_USE_XML_CONFIG, 1377
DomainParticipantConfigParams, 1373
ENTITY_NAME_USE_XML_CONFIG, 1377
participant_name, 1374
participant_qos_library_name, 1374
participant_qos_profile_name, 1375
QOS_ELEMENT_NAME_USE_XML_CONFIG, 1377
rti::flat, 735
rti::flat::AbstractAlignedList< ElementOffset >, 781
begin, 782
end, 782
iterator, 782
rti::flat::AbstractBuilder, 783
~AbstractBuilder, 783
capacity, 784
discard, 784
is_nested, 784
is_valid, 784
rti::flat::AbstractListBuilder< DHeaderGen >, 791
element_count, 791
rti::flat::AbstractPrimitiveList< T >, 792
get_element, 792
set_element, 794
rti::flat::AbstractSequenceBuilder< DHeaderGen >, 794
element_count, 795
rti::flat::AggregationBuilder, 800
rti::flat::FinalAlignedArrayOffset< ElementOffset, N >, 1565
get_element, 1566
rti::flat::FinalArrayOffset< ElementOffset, N >, 1567
get_element, 1568
rti::flat::FinalOffset< T >, 1568
rti::flat::FinalSequenceBuilder< ElementOffset >, 1569
add_n, 1570
add_next, 1570
finish, 1570
rti::flat::flat_type_traits< T >, 1571
rti::flat::MutableArrayBuilder< ElementBuilder, N >, 1753
build_next, 1754
finish, 1754
Offset, 1754
rti::flat::MutableArrayOffset< ElementOffset, N >, 1755
get_element, 1756
rti::flat::MutableOffset, 1756
rti::flat::MutableSequenceBuilder< ElementBuilder >, 1757
build_next, 1758
finish, 1758
Offset, 1758
rti::flat::OffsetBase, 1875
get_buffer, 1877
get_buffer_size, 1877
is_cpp_compatible, 1876
is_null, 1876
operator!=, 1879
operator<, 1878
operator<=, 1878
operator>, 1878
operator>=, 1878
operator==, 1879
rti::flat::PrimitiveArrayOffset< T, N >, 1951
element_count, 1952
rti::flat::PrimitiveConstOffset< T >, 1952
get, 1953
rti::flat::PrimitiveOffset< T >, 1953
set, 1954
rti::flat::PrimitiveSequenceBuilder< T >, 1955
add_n, 1956, 1957
add_next, 1955
finish, 1957

rti::flat::PrimitiveSequenceOffset< T >, 1958
element_count, 1958
rti::flat::Sample< OffsetType >, 2274
clone, 2277
ConstOffset, 2275
create_data, 2276
delete_data, 2277
Offset, 2275
root, 2276
rti::flat::SequenceIterator< E, OffsetKind >, 2328
advance, 2333
difference_type, 2332
is_null, 2332
iterator_category, 2331
operator!=, 2335
operator<, 2334
operator<=, 2334
operator>, 2334
operator>=, 2334
operator*, 2332
operator++, 2333
operator->, 2333
operator==, 2335
pointer, 2331
reference, 2331
SequenceIterator, 2332
value_type, 2331
rti::flat::SequenceOffset< ElementOffset >, 2343
element_count, 2344
get_element, 2344
rti::flat::StringBuilder, 2392
finish, 2393
set_string, 2393
rti::flat::StringOffset, 2394
element_count, 2395
get_string, 2395
rti::flat::UnionBuilder< Discriminator >, 2577
rti::pub, 736
find_datawriter_by_name, 748, 749
find_datawriter_by_topic_name, 747
find_datawriters, 746
find_flow_controller, 750
find_publisher, 744
find_publishers, 743, 744
implicit_publisher, 738
is_matched_subscription_active, 741
matched_subscription_data, 742
matched_subscription_participant_data, 739
matched_subscriptions_locators, 741
rti::pub::AcknowledgmentInfo, 795
response_data, 796
sample_identity, 796
subscription_handle, 796
valid_response_data, 796
rti::pub::FlowController, 1572
close, 1576
closed, 1576
find_flow_controller, 1576
FlowController, 1573
name, 1574
participant, 1574
property, 1574
retain, 1576
trigger_flow, 1575
rti::pub::FlowControllerProperty, 1577
EarliestDeadlineFirst, 1580
FlowControllerProperty, 1578
HighestPriorityFirst, 1580
RoundRobin, 1580
scheduling_policy, 1579
token_bucket, 1579
rti::pub::FlowControllerSchedulingPolicy_def, 1581
EARLIEST_DEADLINE_FIRST, 1582
HIGHEST_PRIORITY_FIRST, 1583
ROUND_ROBIN, 1582
type, 1581
rti::pub::FlowControllerTokenBucketProperty, 1583
bytes_per_token, 1587, 1588
FlowControllerTokenBucketProperty, 1584
max_tokens, 1585
period, 1587
tokens_added_per_period, 1585, 1586
tokens_leaked_per_period, 1586
rti::pub::WriteParams, 2642
cookie, 2647
flag, 2649
handle, 2648
identity, 2646
priority, 2648
related_reader_guid, 2651
related_sample_identity, 2646
related_source_guid, 2651
replace_automatic_values, 2644
reset, 2644
source_guid, 2650
source_timestamp, 2647
WriteParams, 2644
rti::queuing::ConsumerAvailabilityParams, 949
reception_enabled, 950
unacked_threshold, 950
rti::queuing::NoMatchingQueueException, 1835
what, 1836
rti::queuing::NoOpQueueConsumerListener< T >, 1856
on_sample_available, 1857
on_shared_reader_queue_matched, 1857
rti::queuing::NoOpQueueProducerListener< T >, 1858
on_sample_acknowledged, 1858
on_shared_reader_queue_matched, 1858

rti::queuing::NoOpQueueReplierListener< TReq, TRep >, 1859
on_reply_acknowledged, 1860
on_reply_shared_reader_queue_matched, 1861
on_request_available, 1860
on_request_shared_reader_queue_matched, 1860
rti::queuing::NoOpQueueRequesterListener< TReq, TRep >, 1861
on_reply_available, 1862
on_reply_shared_reader_queue_matched, 1863
on_request_acknowledged, 1862
on_request_shared_reader_queue_matched, 1863
rti::queuing::QueueConsumer< T >, 2063
acknowledge_all, 2069
acknowledge_sample, 2069
enable, 2069
get_listener, 2068
guid, 2070
has_matching_reader_queue, 2075
listener, 2067, 2068
QueueConsumer, 2066, 2067
read_samples, 2073
reader, 2070
receive_samples, 2071
send_availability, 2074
set_listener, 2068
take_samples, 2071, 2072
wait_for_samples, 2073
rti::queuing::QueueConsumerListener< T >, 2075
on_sample_available, 2076
on_shared_reader_queue_matched, 2077
rti::queuing::QueueConsumerParams, 2077
enable_availability, 2078
QueueConsumerParams, 2078
rti::queuing::QueueEntityParams< ActualEntity >, 2079
entity_name, 2080
qos_profile, 2079
shared_subscriber_name, 2080
rti::queuing::QueueProducer< T >, 2080
enable, 2085
get_listener, 2084
guid, 2089
has_matching_reader_queue, 2089
listener, 2083, 2084
QueueProducer, 2082, 2083
send_sample, 2085
set_listener, 2084
wait_for_acknowledgments, 2086, 2087
writer, 2088
rti::queuing::QueueProducerListener< T >, 2090
on_sample_acknowledged, 2091
on_shared_reader_queue_matched, 2091
rti::queuing::QueueProducerParams, 2092
enable_sample_replication, 2093
enable_wait_for_ack, 2093
QueueProducerParams, 2093
rti::queuing::QueueReplier< TReq, TRep >, 2094
acknowledge_all, 2100
acknowledge_request, 2100
consumer, 2101
enable, 2098
get_listener, 2098
get_write_params_for_related_request, 2104
guid, 2106
has_matching_reply_reader_queue, 2102
has_matching_request_reader_queue, 2102
listener, 2097
producer, 2102
QueueReplier, 2096
read_requests, 2105, 2106
reader, 2106
receive_requests, 2103, 2104
send_availability, 2102
send_reply, 2099
set_listener, 2098
take_requests, 2105
wait_for_acknowledgments, 2101
wait_for_requests, 2103
writer, 2106
rti::queuing::QueueReplierListener< TReq, TRep >, 2107
on_reply_acknowledged, 2108
on_reply_shared_reader_queue_matched, 2109
on_request_available, 2108
on_request_shared_reader_queue_matched, 2108
rti::queuing::QueueReplierParams, 2109
enable_availability, 2110
enable_sample_replication, 2111
enable_wait_for_ack, 2111
QueueReplierParams, 2110
rti::queuing::QueueRequester< TReq, TRep >, 2111
acknowledge_all, 2121
acknowledge_reply, 2121
consumer, 2122
enable, 2116
get_listener, 2115
guid, 2125
has_matching_reply_reader_queue, 2125
has_matching_request_reader_queue, 2124
listener, 2115
producer, 2122
QueueRequester, 2114
read_replies, 2118, 2119
reader, 2125
receive_replies, 2116, 2117
send_availability, 2124
send_request, 2119, 2120
set_listener, 2116
take_replies, 2117–2119

wait_for_acknowledgments, 2121, 2122
wait_for_replies, 2123
writer, 2125
rti::queuing::QueueRequesterListener< TReq, TRep >, 2126
on_reply_available, 2127
on_reply_shared_reader_queue_matched, 2128
on_request_acknowledged, 2127
on_request_shared_reader_queue_matched, 2127
rti::queuing::QueueRequesterParams, 2128
enable_availability, 2129
enable_sample_replication, 2130
enable_wait_for_ack, 2130
QueueRequesterParams, 2129
rti::request::IsReplyRelatedPredicate< T >, 1622
IsReplyRelatedPredicate, 1623
operator(), 1623, 1624
rti::request::Replier< RequestType, ReplyType >, 2168
get_listener, 2177
listener, 2176, 2177
matched_requester_count, 2178
read_requests, 2176
receive_requests, 2175
Replier, 2171, 2172
reply_datawriter, 2178
request_datareader, 2178
send_reply, 2173, 2174
set_listener, 2177
take_requests, 2176
wait_for_requests, 2174
rti::request::ReplierListener< RequestType, ReplyType >, 2179
on_request_available, 2180
rti::request::ReplierParams, 2180
datareader_qos, 2182
datawriter_qos, 2182
publisher, 2182
ReplierParams, 2181
reply_topic_name, 2182
reply_type, 2183
request_topic_name, 2182
request_type, 2183
service_instance_name, 2183
service_name, 2181
subscriber, 2183
rti::request::Requester< RequestType, ReplyType >, 2187
matched_replier_count, 2198
read_replies, 2194, 2195
receive_replies, 2192, 2193
reply_datareader, 2197
request_datawriter, 2197
Requester, 2189, 2190
send_request, 2191
take_replies, 2193, 2194
wait_for_replies, 2195, 2196
wait_for_service, 2190
rti::request::RequesterParams, 2199
datareader_qos, 2201
datawriter_qos, 2201
publisher, 2201
reply_topic_name, 2201
reply_type, 2202
request_topic_name, 2200
request_type, 2202
RequesterParams, 2200
require_matching_service_on_send_request, 2202
service_name, 2200
subscriber, 2202
rti::request::SimpleReplier< RequestType, ReplyType >, 2368
SimpleReplier, 2369
rti::sub, 751
begin, 765
copy_to_sample, 765
create_topic_query_data_from_service_request, 768
end, 766
find_datareader_by_name, 762, 763
find_datareader_by_topic_description, 763
find_datareader_by_topic_name, 760
find_datareaders, 759, 760
find_subscriber, 758
find_subscribers, 757, 758
find_topic_query, 769
implicit_subscriber, 754
is_matched_publication_alive, 755
matched_publication_data, 756
matched_publication_participant_data, 755
operator<<, 765
operator==, 767
swap, 766
unpack, 769–771
valid_data, 766, 767
rti::sub::AckResponseData, 798
AckResponseData, 799
begin, 799
end, 800
value, 799
rti::sub::cond::DataReaderStatusConditionHandler< T >, 1111
DataReaderStatusConditionHandler, 1112
rti::sub::IsValidData< T >, 1624
operator(), 1625
sample_type, 1625
rti::sub::LoanedSample< T >, 1659
copy_to_sample, 1663
data, 1661
DataType, 1661

info, 1661
InfoType, 1661
operator const DataType &, 1662
operator<<, 1663
operator==, 1662
rti::sub::ManipulatorSelector< T >, 1700
rti::sub::SampleIterator< T >, 2296
operator++, 2296
rti::sub::SampleProcessor, 2305
attach_reader, 2309
detach_reader, 2309
readers, 2310
SampleProcessor, 2307, 2308
rti::sub::SharedSamples< T >, 2367
iterator, 2367
rti::sub::status::DataStateEx, 1126
any, 1131
any_data, 1132
data_state, 1129
DataStateEx, 1128
instance_state, 1131
new_data, 1131
new_instance, 1132
operator!=, 1130
operator<<, 1129
operator>>, 1129
operator==, 1130
sample_state, 1130
stream_kind, 1129
swap, 1132
view_state, 1130, 1131
rti::sub::status::StreamKind, 2390
any, 2392
live, 2392
MaskType, 2391
StreamKind, 2391
topic_query, 2392
rti::sub::TopicQuery, 2516
close, 2518
closed, 2518
datareader, 2518
find_topic_query, 2519
guid, 2518
SelectAll, 2519
TopicQuery, 2517
UseReaderContentFilter, 2518
rti::sub::TopicQueryData, 2520
create_topic_query_data_from_service_request, 2521
original_related_reader_guid, 2521
selection, 2520
topic_name, 2521
rti::sub::TopicQuerySelection, 2526
filter, 2528
Kind, 2526
kind, 2528
TopicQuerySelection, 2527
rti::sub::TopicQuerySelectionKind_def, 2528
CONTINUOUS, 2529
HISTORY_SNAPSHOT, 2529
type, 2529
rti::sub::ValidLoanedSamples< T >, 2595
begin, 2597–2599
const_iterator, 2596
end, 2598, 2599
iterator, 2596
swap, 2600
ValidLoanedSamples, 2597
value_type, 2596
rti::sub::ValidSampleIterator< T >, 2600
rti::test::DisableMonitoringLibraryGuard, 1265
rti::test::EnvVarToken, 1539
rti::test::ParticipantFactoryGuard, 1916
rti::topic, 771
find_registered_content_filters, 775, 776
find_topics, 774
rti::topic::ContentFilter< T, CompileData >, 950
compile, 952
evaluate, 953
finalize, 953
rti::topic::CustomFilter< T >, 967
CustomFilter, 968
get, 969
rti::topic::dynamic_type< TopicType >, 1461
rti::topic::ExpressionProperty, 1547
ExpressionProperty, 1548
key_only_filter, 1548, 1549
writer_side_filter_optimization, 1549
rti::topic::extensibility< TopicType >, 1550
rti::topic::FilterSampleInfo, 1562
priority, 1565
related_reader_guid, 1564
related_sample_identity, 1563
related_source_guid, 1563
sample_identity, 1564
source_guid, 1564
rti::topic::no_compile_data_t, 1835
rti::topic::PrintFormatKind_def, 1961
DEFAULT, 1962
JSON, 1962
type, 1961
XML, 1962
rti::topic::PrintFormatProperty, 1962
Default, 1966
enum_as_int, 1964, 1965
include_root_elements, 1965
Json, 1966
kind, 1963

pretty_print, 1964
PrintFormatProperty, 1963
Xml, 1966
rti::topic::ServiceRequest, 2358
instance_id, 2359
request_body, 2359
service_id, 2359
rti::topic::topic_type_disabled_copy< TopicType >, 2492
rti::topic::topic_type_has_external_members< TopicType >, 2492
rti::topic::trust::EndpointTrustAlgorithmInfo, 1516
EndpointTrustAlgorithmInfo, 1517
interceptor, 1517
rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo, 1517
EndpointTrustInterceptorAlgorithmInfo, 1518
required_mask, 1518
supported_mask, 1518
rti::topic::trust::EndpointTrustProtectionInfo, 1519
bitmask, 1519
EndpointTrustProtectionInfo, 1519
plugin_bitmask, 1520
rti::topic::trust::ParticipantTrustAlgorithmInfo, 1916
interceptor, 1917
key_establishment, 1917
ParticipantTrustAlgorithmInfo, 1916
signature, 1917
rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo, 1917
builtin_endpoints_required_mask, 1918
builtin_kx_endpoints_required_mask, 1919
ParticipantTrustInterceptorAlgorithmInfo, 1918
supported_mask, 1918
user_endpoints_default_required_mask, 1919
rti::topic::trust::ParticipantTrustKeyEstablishmentAlgorithmInfo, 1919
ParticipantTrustKeyEstablishmentAlgorithmInfo, 1920
shared_secret, 1920
rti::topic::trust::ParticipantTrustProtectionInfo, 1920
bitmask, 1921
ParticipantTrustProtectionInfo, 1921
plugin_bitmask, 1921
rti::topic::trust::ParticipantTrustSignatureAlgorithmInfo, 1922
message_auth, 1923
ParticipantTrustSignatureAlgorithmInfo, 1922
trust_chain, 1923
rti::topic::trust::TrustAlgorithmRequirements, 2562
required_mask, 2563
supported_mask, 2562
TrustAlgorithmRequirements, 2562
rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >, 2652
writer_attach, 2656
writer_compile, 2654
writer_detach, 2656
writer_evaluate, 2655
writer_finalize, 2655
writer_return_loan, 2656
rti::topic::WriterContentFilterHelper< T, CompileData, WriterFilterData >, 2657
add_cookie, 2659
writer_evaluate_helper, 2658
rti::util, 777
rti::util::activation, 778
rti::util::atomic_shared_ptr< T >, 862
rti::util::discovery, 778
rti::util::function_history, 778
rti::util::heap_monitoring, 779
rti::util::heap_monitoring::HeapMonitoringParams, 1599
HeapMonitoringParams, 1600
snapshot_content_format, 1601
snapshot_output_format, 1600, 1601
rti::util::heap_monitoring::SnapshotContentFormat_def, 2370
ACTIVITY, 2371
DEFAULT, 2371
FUNCTION, 2371
MINIMAL, 2371
TOPIC, 2371
type, 2370
rti::util::heap_monitoring::SnapshotOutputFormat_def, 2371
COMPRESSED, 2372
STANDARD, 2372
type, 2371
rti::util::network_capture, 780
rti::util::network_capture::ContentKindMask, 962
all, 964
ContentKindMask, 963
default_mask, 964
encrypted, 964
MaskType, 963
none, 964
user, 964
rti::util::network_capture::NetworkCaptureParams, 1829
checkpoint_thread_settings, 1833
dropped_content, 1830, 1831
frame_queue_size, 1834
parse_encrypted_content, 1832
traffic, 1831
transports, 1830
rti::util::network_capture::TrafficKindMask, 2529
all, 2532
default_mask, 2532
in, 2532
MaskType, 2530
none, 2532

out, 2531
TrafficKindMask, 2531
rti::util::StreamFlagSaver, 2390
RTI_MONITORING_EVENT_TOPIC_NAME
Monitoring Library 2.0, 600
RTI_Monitoring_initialize
Monitoring Library 2.0, 599
RTI_MONITORING_LOGGING_TOPIC_NAME
Monitoring Library 2.0, 600
RTI_MONITORING_PERIODIC_TOPIC_NAME
Monitoring Library 2.0, 600
rtps_app_id
rti::core::policy::WireProtocol, 2636, 2637
RTPS_AUTO_ID
rti::core::policy::WireProtocol, 2641
RTPS_AUTO_ID_FROM_IP
rti::core::policy::WireProtocolAutoKind_def, 2642
RTPS_AUTO_ID_FROM_MAC
rti::core::policy::WireProtocolAutoKind_def, 2642
RTPS_AUTO_ID_FROM_UUID
rti::core::policy::WireProtocolAutoKind_def, 2642
rtps_auto_id_kind
rti::core::policy::WireProtocol, 2639
rtps_host_id
rti::core::policy::WireProtocol, 2636
rtps_instance_id
rti::core::policy::WireProtocol, 2637, 2638
rtps_object_id
rti::core::policy::DataReaderProtocol, 1061
rti::core::policy::DataWriterProtocol, 1212, 1213
rtps_port
NDDS_Transport_UDP_WAN_CommPortsMappingInfo, 1797
rtps_protocol_version
dds::topic::ParticipantBuiltinTopicData, 1910
dds::topic::PublicationBuiltinTopicData, 1987
dds::topic::SubscriptionBuiltinTopicData, 2436
rtps_reliable_reader
rti::core::policy::DataReaderProtocol, 1064, 1065
rtps_reliable_writer
rti::core::policy::DataWriterProtocol, 1217, 1218
rtps_reserved_port_mask
rti::core::policy::WireProtocol, 2639
rtps_vendor_id
dds::topic::ParticipantBuiltinTopicData, 1911
dds::topic::PublicationBuiltinTopicData, 1988
dds::topic::SubscriptionBuiltinTopicData, 2437
rtps_well_known_ports
rti::core::policy::WireProtocol, 2638
RtpsReliableReaderProtocol
rti::core::RtpsReliableReaderProtocol, 2223
RtpsReliableWriterProtocol
rti::core::policy::RtpsReliableWriterProtocol, 2232
RtpsReservedPortKindMask
rti::core::policy::RtpsReservedPortKindMask, 2257
RtpsWellKnownPorts
rti::core::RtpsWellKnownPorts, 2261
run
dds::rpc::Server, 2347

Safe Enumeration, 429
safe_enum
dds::core::safe_enum< def, inner >, 2267
Sample
dds::sub::Sample< T >, 2271, 2272
sample
dds::sub::Rank, 2132
dds::topic::TopicInstance< T >, 2507
Sample Information, 71
Sample Memory Management QoS, 230
Sample Ordering, 234
sample_count
rti::core::status::DataReaderCacheStatus, 1048
rti::core::status::DataWriterCacheStatus, 1200
sample_count_peak
rti::core::status::DataReaderCacheStatus, 1048
rti::core::status::DataWriterCacheStatus, 1200
sample_identity
rti::pub::AcknowledgmentInfo, 796
rti::topic::FilterSampleInfo, 1564
sample_lost
dds::core::status::StatusMask, 2381
sample_lost_status
dds::sub::DataReader< T >, 1011
sample_rejected
dds::core::status::StatusMask, 2382
sample_rejected_status
dds::sub::DataReader< T >, 1011
sample_removed
dds::core::status::StatusMask, 2389
sample_state
dds::sub::status::DataState, 1123
rti::sub::status::DataStateEx, 1130
sample_type
rti::sub::IsValidData< T >, 1625
SampleFlag
rti::core::SampleFlag, 2279, 2280
SampleIdentity
rti::core::SampleIdentity, 2283
SampleLostState
rti::core::status::SampleLostState, 2298
SampleProcessor, 566
rti::sub::SampleProcessor, 2307, 2308
SampleRejectedState
dds::core::status::SampleRejectedState, 2312
samples_per_app_ack
rti::core::RtpsReliableReaderProtocol, 2228, 2229
samples_per_period

rti::core::policy::TopicQueryDispatch, 2525
samples_per_virtual_heartbeat
rti::core::policy::RtpsReliableWriterProtocol, 2238
SampleState
dds::sub::status::SampleState, 2317
scheduling_policy
rti::pub::FlowControllerProperty, 1579
scope
dds::core::policy::DestinationOrder, 1262
ScopedLoggerVerbosity
rti::config::ScopedLoggerVerbosity, 2319
SDP
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask, 1325
SDP2
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask, 1325
sec
dds::core::Duration, 1452
dds::core::Time, 2460, 2461
secure_volatile_reader
rti::core::policy::DiscoveryConfig, 1309, 1310
secure_volatile_writer
rti::core::policy::DiscoveryConfig, 1307, 1308
secure_volatile_writer_publish_mode
rti::core::policy::DiscoveryConfig, 1309
Security, 235
security
rti::config::LogCategory_def, 1683
security_event
Logging, 459
security_event_forwarding_level
rti::core::MonitoringLoggingForwardingSettings, 1734, 1735
SEDP
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask, 1324
select
dds::sub::DataReader< T >, 1003
select_default_discriminator
rti::config::compliance::XTypesMask, 2671
SelectAll
rti::sub::TopicQuery, 2519
selection
rti::sub::TopicQueryData, 2520
Selector
dds::sub::DataReader< T >::Selector, 2321
send_availability
rti::queuing::QueueConsumer< T >, 2074
rti::queuing::QueueReplier< TReq, TRep >, 2102
rti::queuing::QueueRequester< TReq, TRep >, 2124
send_blocking
NDDS_Transport_UDPv4_Property_t, 1804
NDDS_Transport_UDPv4_WAN_Property_t, 1813
NDDS_Transport_UDPv6_Property_t, 1824
send_ping
NDDS_Transport_UDPv4_Property_t, 1805
NDDS_Transport_UDPv4_WAN_Property_t, 1814
NDDS_Transport_UDPv6_Property_t, 1825
send_reply
rti::queuing::QueueReplier< TReq, TRep >, 2099
rti::request::Replier< RequestType, ReplyType >, 2173, 2174
send_request
rti::queuing::QueueRequester< TReq, TRep >, 2119, 2120
rti::request::Requester< RequestType, ReplyType >, 2191
send_sample
rti::queuing::QueueProducer< T >, 2085
send_socket_buffer_size
NDDS_Transport_UDPv4_Property_t, 1800
NDDS_Transport_UDPv4_WAN_Property_t, 1810
NDDS_Transport_UDPv6_Property_t, 1820
send_window_decrease_factor
rti::core::policy::RtpsReliableWriterProtocol, 2253
send_window_increase_factor
rti::core::policy::RtpsReliableWriterProtocol, 2252
send_window_size
rti::core::status::DataWriterProtocolStatus, 1223
send_window_update_period
rti::core::policy::RtpsReliableWriterProtocol, 2251, 2252
Sending Large Data, 197
sent_ack_bytes
rti::core::status::DataReaderProtocolStatus, 1069
sent_ack_count
rti::core::status::DataReaderProtocolStatus, 1069
sent_gap_bytes
rti::core::status::DataWriterProtocolStatus, 1223
sent_gap_count
rti::core::status::DataWriterProtocolStatus, 1223
sent_heartbeat_bytes
rti::core::status::DataWriterProtocolStatus, 1221
sent_heartbeat_count
rti::core::status::DataWriterProtocolStatus, 1221
sent_nack_bytes
rti::core::status::DataReaderProtocolStatus, 1069
sent_nack_count
rti::core::status::DataReaderProtocolStatus, 1069
sent_nack_fragment_bytes
rti::core::status::DataReaderProtocolStatus, 1072
sent_nack_fragment_count
rti::core::status::DataReaderProtocolStatus, 1071
sentinel_in_empty_union
rti::config::compliance::XTypesMask, 2671
sequence
Supporting Types and Constants, 436

sequence_number
rti::core::SampleIdentity, 2284
SEQUENCE_TYPE
dds::core::xtypes::TypeKind_def, 2573
SequenceIterator
rti::flat::SequenceIterator< E, OffsetKind >, 2332
SequenceNumber
rti::core::SequenceNumber, 2337, 2338
SequenceType
dds::core::xtypes::SequenceType, 2345, 2346
serialize_key_with_dispose
rti::core::policy::DataWriterProtocol, 1215, 1216
serialized_type_object_dynamic_allocation_threshold
rti::core::policy::DomainParticipantResourceLimits, 1421, 1422
Server
dds::rpc::Server, 2347
Server-side API, 408
RobotControlService, 408
ServiceParams, 408
SERVICE, 546
ServiceKind, 546
Service
rti::core::policy::Service, 2351, 2352
service
dds::topic::PublicationBuiltinTopicData, 1990
dds::topic::SubscriptionBuiltinTopicData, 2438
Logging, 459
service_cleanup_delay
dds::core::policy::DurabilityService, 1445
service_forwarding_level
rti::core::MonitoringLoggingForwardingSettings, 1735
service_id
rti::core::status::ServiceRequestAcceptedStatus, 2361
rti::topic::ServiceRequest, 2359
service_instance_name
rti::request::ReplierParams, 2183
service_name
rti::request::ReplierParams, 2181
rti::request::RequesterParams, 2200
service_request
rti::core::policy::DiscoveryConfigBuiltinChannelKindMask, 1321
service_request_accepted
dds::core::status::StatusMask, 2388
service_request_accepted_status
dds::pub::DataWriter< T >, 1178
service_request_reader
rti::core::policy::DiscoveryConfig, 1304, 1305
service_request_topic_name
ServiceRequest Built-in Topic, 568
service_request_writer
rti::core::policy::DiscoveryConfig, 1301, 1302
service_request_writer_data_lifecycle
rti::core::policy::DiscoveryConfig, 1303
service_request_writer_publish_mode
rti::core::policy::DiscoveryConfig, 1303, 1304
ServiceEndpoint
dds::rpc::ServiceEndpoint< Dispatcher >, 2356
ServiceKind
SERVICE, 546
ServiceParams
Server-side API, 408
ServiceRequest Built-in Topic, 566
service_request_topic_name, 568
ServiceRequestId, 567
ServiceRequestId
ServiceRequest Built-in Topic, 567
set
dds::core::optional< T >, 1884
dds::core::policy::DataTag, 1135, 1136
rti::core::pointer< T >, 1939
rti::core::policy::Property, 1973–1975
rti::flat::PrimitiveOffset< T >, 1954
set_attribute_mask
Activity Context, 450
set_cdr_buffer
dds::core::xtypes::DynamicData, 1482
set_default_params
Network Capture, 582
set_discriminator
dds::core::xtypes::DynamicData, 1478
set_element
rti::flat::AbstractPrimitiveList< T >, 794
set_listener
dds::domain::DomainParticipant, 1335, 1336
dds::pub::DataWriter< T >, 1169, 1170
dds::pub::Publisher, 1999, 2000
dds::sub::DataReader< T >, 1006, 1007
dds::sub::Subscriber, 2415, 2416
dds::topic::Topic< T >, 2482, 2483
rti::queuing::QueueConsumer< T >, 2068
rti::queuing::QueueProducer< T >, 2084
rti::queuing::QueueReplier< TReq, TRep >, 2098
rti::queuing::QueueRequester< TReq, TRep >, 2116
rti::request::Replier< RequestType, ReplyType >, 2177
set_rpc_mask
Compliance Configuration, 454
set_string
rti::flat::StringBuilder, 2393
set_tuple
dds::core::xtypes::DynamicData, 1493
rti::core::xtypes, 724
set_values
dds::core::xtypes::DynamicData, 1474

set_xtypes_mask
Compliance Configuration, 452
settings
rti::core::policy::TransportMulticast, 2545
rti::core::policy::TransportUnicast, 2558, 2559
SHARED
dds::core::policy::OwnershipKind_def, 1906
Shared
dds::core::policy::Ownership, 1905
Shared Memory Transport, 380
NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT, 384
NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT, 384
NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_1424, 385
NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT, 384
NDDS_Transport_Shmem_new, 385
NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT, 384
NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT, 385
NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT, 385
NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT, 384
shared_ptr
Supporting Types and Constants, 435
shared_secret
rti::topic::trust::ParticipantTrustKeyEstablishmentAlgorithmInfo, 1920
shared_subscriber_name
rti::queuing::QueueEntityParams< ActualEntity >, 2080
SharedSamples
dds::sub::SharedSamples< T, DELEGATE >, 2363
SHMEM
rti::core::LocatorKind_def, 1681
rti::core::TransportClassId_def, 2541
Shmem
rti::core::policy::TransportBuiltin, 2535
shmem
rti::core::policy::TransportBuiltinMask, 2539
SHMEM_510
rti::core::TransportClassId_def, 2541
SHMEM_ALIAS
rti::core::policy::TransportBuiltin, 2536
shmem_ref_settings
rti::core::policy::DataWriterTransferMode, 1254, 1255
shmem_ref_transfer_mode_attached_segment_allocation
rti::core::policy::DataReaderResourceLimits, 1106
shmem_ref_transfer_mode_max_segments
rti::core::policy::DomainParticipantResourceLimits, 1433
ShmemRefSettings
rti::core::policy::DataWriterTransferMode, 1254
shrink_to_fit
rti::core::bounded_sequence< T, MaxLength >, 902
shutdown_cleanup_period
rti::core::policy::Database, 973
shutdown_timeout
rti::core::policy::Database, 972
signature
rti::topic::trust::ParticipantTrustAlgorithmInfo, 1917
silent
Logging, 458
rti::config::Verbosity_def, 2614
SimpleReplier
rti::request::SimpleReplier< RequestType, ReplyType >, 2369
size
dds::core::basic_string< CharType, Allocator >, 877
dds::core::policy::DataTag, 1137
dds::core::vector< T >, 2607
rti::core::bounded_sequence< T, MaxLength >, 901
rti::core::policy::Property, 1974
size_type
rti::core::bounded_sequence< T, MaxLength >, 891
skip_deserialization_MAX_DEFAULT, 1502, 1503
sleep
Other Utilities, 596
snapshot_content_format
rti::util::heap_monitoring::HeapMonitoringParams, 1601
snapshot_output_format
rti::util::heap_monitoring::HeapMonitoringParams, 1600, 1601
SnapshotContentFormat
Heap Monitoring, 590
SnapshotOutputFormat
Heap Monitoring, 590
snippet_compatibility_5_1_0_transport_enable
Builtin Qos Profiles, 506
snippet_compatibility_connext_micro_version_2_4_3
Builtin Qos Profiles, 506
snippet_compatibility_other_dds_vendors_enable
Builtin Qos Profiles, 506
snippet_feature_auto_tuning_enable
Builtin Qos Profiles, 500
snippet_feature_monitoring2_enable
Builtin Qos Profiles, 501
snippet_feature_monitoring_enable
Builtin Qos Profiles, 501
snippet_feature_security_enable
Builtin Qos Profiles, 502

snippet_feature_topic_query_enable
Builtin Qos Profiles, 502
snippet_feature_type_support_v1_disable
Builtin Qos Profiles, 503
snippet_feature_type_support_v1_enable
Builtin Qos Profiles, 502
snippet_feature_type_support_v2_disable
Builtin Qos Profiles, 503
snippet_feature_type_support_v2_enable
Builtin Qos Profiles, 503
snippet_optimization_discovery_common
Builtin Qos Profiles, 494
snippet_optimization_discovery_endpoint_fast
Builtin Qos Profiles, 495
snippet_optimization_discovery_participant_compact
Builtin Qos Profiles, 495
snippet_optimization_reliability_protocol_common
Builtin Qos Profiles, 491
snippet_optimization_reliability_protocol_dynamicmemalloc
Builtin Qos Profiles, 494
snippet_optimization_reliability_protocol_high_rate
Builtin Qos Profiles, 492
snippet_optimization_reliability_protocol_keep_all
Builtin Qos Profiles, 491
snippet_optimization_reliability_protocol_keep_last
Builtin Qos Profiles, 492
snippet_optimization_reliability_protocol_large_data
Builtin Qos Profiles, 493
snippet_optimization_reliability_protocol_low_latency
Builtin Qos Profiles, 493
snippet_optimization_transport_large_buffers
Builtin Qos Profiles, 495
snippet_qos_policy_batching_enable
Builtin Qos Profiles, 499
snippet_qos_policy_durability_persistent
Builtin Qos Profiles, 498
snippet_qos_policy_durability_transient
Builtin Qos Profiles, 498
snippet_qos_policy_durability_transient_local
Builtin Qos Profiles, 498
snippet_qos_policy_flow_controller_209mbps
Builtin Qos Profiles, 499
snippet_qos_policy_flow_controller_52mbps
Builtin Qos Profiles, 500
snippet_qos_policy_flow_controller_838mbps
Builtin Qos Profiles, 499
snippet_qos_policy_history_keep_all
Builtin Qos Profiles, 497
snippet_qos_policy_history_keep_last_1
Builtin Qos Profiles, 497
snippet_qos_policy_publish_mode_asynchronous
Builtin Qos Profiles, 497
snippet_qos_policy_reliability_best_effort
Builtin Qos Profiles, 496
snippet_qos_policy_reliability_reliable
Builtin Qos Profiles, 496
snippet_transport_tcp_lan_client
Builtin Qos Profiles, 504
snippet_transport_tcp_wan_asymmetric_client
Builtin Qos Profiles, 505
snippet_transport_tcp_wan_asymmetric_server
Builtin Qos Profiles, 504
snippet_transport_tcp_wan_symmetric_client
Builtin Qos Profiles, 504
snippet_transport_udp_avoid_ip_fragmentation
Builtin Qos Profiles, 505
snippet_transport_udp_wan
Builtin Qos Profiles, 505
source_guid
dds::sub::SampleInfo, 2293
rti::pub::WriteParams, 2650
rti::topic::FilterSampleInfo, 1564
source_timestamp
dds::sub::SampleInfo, 2289
rti::pub::WriteParams, 2647
source_timestamp_resolution
rti::core::policy::Batch, 885, 886
source_timestamp_tolerance
dds::core::policy::DestinationOrder, 1262, 1263
SourceTimestamp
dds::core::policy::DestinationOrder, 1261
SPDP
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask, 1324
SPDP2
rti::core::policy::DiscoveryConfigBuiltinPluginKindMask, 1324
spin
Other Utilities, 597
spin_per_microsecond
Other Utilities, 597
sql_filter_name
Topics, 49
stack_size
rti::core::ThreadSettings, 2451
STANDARD
rti::util::heap_monitoring::SnapshotOutputFormat_def, 2372
start
Network Capture, 583–585
rti::core::cond::AsyncWaitSet, 845
state
dds::sub, 664
dds::sub::DataReader< T >, 1027
dds::sub::DataReader< T >::Selector, 2323
dds::sub::SampleInfo, 2289
state_filter
dds::sub::cond::ReadCondition, 2135

status
NDDS_Transport_Interface_t, 1786
Status Kinds, 429
STATUS_ALL
rti::config::LogLevel_def, 1693
status_all
rti::config::Verbosity_def, 2614
status_changes
dds::core::Entity, 1525
STATUS_LOCAL
rti::config::LogLevel_def, 1693
status_local
rti::config::Verbosity_def, 2614
STATUS_REMOTE
rti::config::LogLevel_def, 1693
status_remote
rti::config::Verbosity_def, 2614
StatusCondition
dds::core::cond::StatusCondition, 2373
StatusMask
dds::core::status::StatusMask, 2378
stdio
rti::core::ThreadSettingsKindMask, 2456
stop
Network Capture, 586
rti::core::cond::AsyncWaitSet, 846, 847
storage_settings
dds::core::policy::Durability, 1440, 1441
stored_size
rti::core::xtypes::DynamicDataInfo, 1495
stream_kind
rti::sub::status::DataStateEx, 1129
StreamKind
rti::sub::status::StreamKind, 2391
string
Supporting Types and Constants, 435
string_profile
rti::core::QosProviderParams, 2050
STRING_TYPE
dds::core::xtypes::TypeKind_def, 2574
string_view
Supporting Types and Constants, 437
stringmatch_filter_name
Topics, 49
StringSeq
Supporting Types and Constants, 436
StringTopicType
dds::core::StringTopicType, 2396, 2397
StringType
dds::core::xtypes::StringType, 2400
StructType
dds::core::xtypes::StructType, 2402–2404, 2406, 2407
STRUCTURE_TYPE
dds::core::xtypes::TypeKind_def, 2574
Subscriber
dds::sub::Subscriber, 2413, 2414
subscriber
dds::sub::AnyDataReader, 810
dds::sub::DataReader< T >, 1005
rti::request::ReplierParams, 2183
rti::request::RequesterParams, 2202
Subscriber Use Cases, 315
subscriber_group_data_max_length
rti::core::policy::DomainParticipantResourceLimits, 1415
subscriber_key
dds::topic::SubscriptionBuiltinTopicData, 2435
subscriber_qos
dds::core::QosProvider, 2032, 2033
Subscribers, 63
Subscription Built-in Topics, 443
subscription_topic_name, 444
Subscription Example, 305
Subscription Module, 61
subscription_handle
rti::pub::AcknowledgmentInfo, 796
subscription_matched
dds::core::status::StatusMask, 2384
subscription_matched_status
dds::sub::DataReader< T >, 1012
subscription_name
dds::topic::SubscriptionBuiltinTopicData, 2437
subscription_reader
rti::core::policy::DiscoveryConfig, 1288
subscription_reader_resource_limits
rti::core::policy::DiscoveryConfig, 1289
subscription_topic_name
Subscription Built-in Topics, 444
subscription_writer
rti::core::policy::DiscoveryConfig, 1291, 1292
subscription_writer_data_lifecycle
rti::core::policy::DiscoveryConfig, 1293
subscription_writer_publish_mode
rti::core::policy::DiscoveryConfig, 1298, 1299
supported_mask
rti::topic::trust::EndpointTrustInterceptorAlgorithmInfo, 1518
rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo, 1918
rti::topic::trust::TrustAlgorithmRequirements, 2562
Supporting Types and Constants, 433
bounded_sequence, 436
ByteSeq, 435
length_auto, 438
LENGTH_UNLIMITED, 438
null, 438
null_type, 436

optional, 436
qos_print_all, 439
sequence, 436
shared_ptr, 435
string, 435
string_view, 437
StringSeq, 436
unregister_thread, 437
wstring, 435
wstring_view, 437
SuspendedPublication
dds::pub::SuspendedPublication, 2442
swap
dds::core, 622
dds::core::Duration, 1460
dds::core::external< T >, 1558
dds::core::optional< T >, 1889
dds::core::safe_enum< def, inner >, 2269
dds::core::vector< T >, 2611
dds::pub::qos, 656
dds::pub::qos::PublisherQos, 2010
dds::sub, 680
dds::sub::LoanedSamples< T >, 1668, 1670
dds::sub::status::DataState, 1125
rti::core::bounded_sequence< T, MaxLength >, 905
rti::core::optional_value< T >, 1898
rti::sub, 766
rti::sub::status::DataStateEx, 1132
rti::sub::ValidLoanedSamples< T >, 2600
synchronization_kind
rti::core::PersistentStorageSettings, 1933
SYNCHRONOUS
rti::core::policy::PublishModeKind_def, 2019
Synchronous
rti::core::policy::PublishMode, 2015
SyslogLevel
Logging, 457
SyslogVerbosity
Logging, 458
System Properties, 301
SYSTEM_RESOURCE_LIMITS, 547
SystemResourceLimits
rti::core::policy::SystemResourceLimits, 2444

take
dds::sub, 662
dds::sub::DataReader< T >, 997, 1001, 1002, 1020, 1021, 1024
dds::sub::DataReader< T >::Selector, 2326–2328
take_noexcept
dds::sub::DataReader< T >, 1018
take_replies
rti::queuing::QueueRequester< TReq, TRep >, 2117–2119
rti::request::Requester< RequestType, ReplyType >, 2193, 2194
take_requests
rti::queuing::QueueReplier< TReq, TRep >, 2105
rti::request::Replier< RequestType, ReplyType >, 2176
take_samples
rti::queuing::QueueConsumer< T >, 2071, 2072
take_snapshot
Discovery Snapshot, 573–577
Heap Monitoring, 593
TBuiltinTopicKey
dds::topic::BuiltinTopicKey, 906
TCPV4_LAN
rti::core::TransportClassId_def, 2541
TCPV4_WAN
rti::core::TransportClassId_def, 2541
telemetry_data
rti::core::policy::Monitoring, 1711
TEntityFactory
dds::core::policy::EntityFactory, 1528, 1529
text
rti::config::LogMessage, 1694
thread
rti::core::MonitoringEventDistributionSettings, 1726
rti::core::MonitoringLoggingDistributionSettings, 1731, 1732
rti::core::MonitoringPeriodicDistributionSettings, 1743
rti::core::policy::AsynchronousPublisher, 835, 836
rti::core::policy::Database, 971, 972
rti::core::policy::Event, 1542, 1543
rti::core::policy::ReceiverPool, 2146, 2147
thread_name_prefix
NDDS_Transport_Property_t, 1793
rti::core::cond::AsyncWaitSetProperty, 861
thread_pool_size
dds::rpc::ServerParams, 2349
rti::core::cond::AsyncWaitSetProperty, 862
thread_safe_write
rti::core::policy::Batch, 886
thread_settings
rti::core::cond::AsyncWaitSetProperty, 861
ThreadSettings
rti::core::ThreadSettings, 2450
ThreadSettingsCpuRotationKind
rti::core, 706
ThreadSettingsKindMask
rti::core::ThreadSettingsKindMask, 2455
Throughput, 236
throw_if_error
rti::core::Result< T >, 2213
Time
dds::core::Time, 2458

Time Support, 427
TIME_BASED_FILTER, 547
time_based_filter
dds::topic::SubscriptionBuiltinTopicData, 2433
time_based_filter_dropped_sample_count
rti::core::status::DataReaderCacheStatus, 1049
time_until_license_expiration
dds::domain::DomainParticipant, 1359
TimeBasedFilter
dds::core::policy::TimeBasedFilter, 2471, 2472
timestamp
rti::config::LogMessage, 1695
TIMESTAMPED
rti::config::PrintFormat_def, 1961
TLSV4_LAN
rti::core::TransportClassId_def, 2541
TLSV4_WAN
rti::core::TransportClassId_def, 2541
to_cdr_buffer
dds::core::xtypes::DynamicData, 1490
rti::core::xtypes, 721
Topic-type serialization and deserialization, 570
to_chrono
dds::core::Duration, 1458
to_microsecs
dds::core::Duration, 1458
dds::core::Time, 2466
to_millisecs
dds::core::Duration, 1458
dds::core::Time, 2466
to_nanosecs
dds::core::Duration, 1457
dds::core::Time, 2467
to_pointer
rti::core::Cookie, 966
to_secs
dds::core::Duration, 1458
dds::core::Time, 2467
to_std_string
dds::core::basic_string< CharType, Allocator >, 879
to_string
dds::core::xtypes::DynamicType, 1508
dds::domain::qos, 641–644
dds::domain::qos::DomainParticipantFactoryQos, 1379–1381
dds::domain::qos::DomainParticipantQos, 1388–1390
dds::pub::qos, 654–657
dds::pub::qos::DataWriterQos, 1233, 1234
dds::pub::qos::PublisherQos, 2011, 2012
dds::sub::qos, 681, 683–686
dds::sub::qos::DataReaderQos, 1079, 1080
dds::sub::qos::SubscriberQos, 2425–2427
dds::topic::qos, 696, 697
dds::topic::qos::TopicQos, 2513–2515
rti::core::ProductVersion, 1968
rti::core::xtypes, 726
Topic traits and data-type support, 445, 447
token_bucket
rti::pub::FlowControllerProperty, 1579
tokens_added_per_period
rti::pub::FlowControllerTokenBucketProperty, 1585, 1586
tokens_leaked_per_period
rti::pub::FlowControllerTokenBucketProperty, 1586
tolerance_source_timestamp_dropped_sample_count
rti::core::status::DataReaderCacheStatus, 1048
TOPIC
dds::core::policy::PresentationAccessScopeKind_def, 1951
rti::core::policy::DestinationOrderScopeKind_def, 1265
rti::util::heap_monitoring::SnapshotContentFormat_def, 2371
Topic
dds::topic::Topic< T >, 2477–2481
topic
dds::pub::DataWriter< T >, 1168
dds::topic::ContentFilteredTopic< T >, 957
rti::config::activity_context::AttributeKindMask, 865
Topic Built-in Topics, 442
topic_topic_name, 442
Topic Module, 47
Topic Queries, 67, 239
TopicQuerySelectionKind, 71
Topic traits and data-type support, 444
from_string, 447
PrintFormatKind, 445
to_string, 445, 447
Topic Use Cases, 308
Topic-type serialization and deserialization, 568
from_cdr_buffer, 569
from_cdr_buffer_no_alloc, 568
to_cdr_buffer, 570
TOPIC_DATA, 548
topic_data
dds::topic::PublicationBuiltinTopicData, 1984
dds::topic::SubscriptionBuiltinTopicData, 2433
dds::topic::TopicBuiltinTopicData, 2499
topic_data_max_length
rti::core::policy::DomainParticipantResourceLimits, 1414
topic_description
dds::sub::DataReader< T >, 1005
topic_expression
rti::core::policy::MulticastMapping, 1748, 1749
topic_name
dds::pub::AnyDataWriter, 817

dds::sub::AnyDataReader, 809
dds::sub::DataReader< T >, 1023
dds::topic::PublicationBuiltinTopicData, 1980
dds::topic::SubscriptionBuiltinTopicData, 2430
rti::sub::TopicQueryData, 2521
topic_qos
dds::core::QosProvider, 2032
topic_qos_w_topic_name
dds::core::QosProvider, 2038
TOPIC_QUERY
rti::core::ServiceRequestId_def, 2362
topic_query
rti::sub::status::StreamKind, 2392
TOPIC_QUERY_DISPATCH, 548
topic_query_guid
dds::sub::SampleInfo, 2294
topic_query_publication_thread
rti::core::policy::AsynchronousPublisher, 838, 839
topic_topic_name
Topic Built-in Topics, 442
TopicAccessScope
dds::core::policy::Presentation, 1949
TopicData
dds::core::policy::TopicData, 2501, 2502
TopicInstance
dds::topic::TopicInstance< T >, 2506
TopicQos
dds::topic::qos::TopicQos, 2511
TopicQuery
rti::sub::TopicQuery, 2517
TopicQueryDispatch
rti::core::policy::TopicQueryDispatch, 2524
TopicQuerySelection
rti::sub::TopicQuerySelection, 2527
TopicQuerySelectionKind
Topic Queries, 71
Topics, 48
sql_filter_name, 49
stringmatch_filter_name, 49
total
rti::core::status::EventCount< IntegerType >, 1545
total_count
dds::core::status::InconsistentTopicStatus, 1612
dds::core::status::LivelinessLostStatus, 1656
dds::core::status::OfferedDeadlineMissedStatus, 1872
dds::core::status::OfferedIncompatibleQosStatus, 1873
dds::core::status::PublicationMatchedStatus, 1991
dds::core::status::RequestedDeadlineMissedStatus, 2184
dds::core::status::RequestedIncompatibleQosStatus, 2186
dds::core::status::SampleLostStatus, 2304
dds::core::status::SampleRejectedStatus, 2315
dds::core::status::SubscriptionMatchedStatus, 2440
rti::core::status::ServiceRequestAcceptedStatus, 2360
total_count_change
dds::core::status::InconsistentTopicStatus, 1612
dds::core::status::LivelinessLostStatus, 1656
dds::core::status::OfferedDeadlineMissedStatus, 1872
dds::core::status::OfferedIncompatibleQosStatus, 1874
dds::core::status::PublicationMatchedStatus, 1991
dds::core::status::RequestedDeadlineMissedStatus, 2184
dds::core::status::RequestedIncompatibleQosStatus, 2186
dds::core::status::SampleLostStatus, 2304
dds::core::status::SampleRejectedStatus, 2315
dds::core::status::SubscriptionMatchedStatus, 2440
total_element_count
dds::core::xtypes::ArrayType, 832
total_samples_dropped_by_instance_replacement
rti::core::status::DataReaderCacheStatus, 1051
TParticipantBuiltinTopicData
dds::topic::ParticipantBuiltinTopicData, 1910
TPartition
dds::core::policy::Partition, 1925, 1926
TPublicationBuiltinTopicData
dds::topic::PublicationBuiltinTopicData, 1979
trace_file_name
rti::core::PersistentStorageSettings, 1931, 1932
traffic
rti::util::network_capture::NetworkCaptureParams, 1831
TrafficKindMask
rti::util::network_capture::TrafficKindMask, 2531
TRANSIENT
dds::core::policy::DurabilityKind_def, 1442
Transient
dds::core::policy::Durability, 1438
TRANSIENT_LOCAL
dds::core::policy::DurabilityKind_def, 1442
TransientLocal
dds::core::policy::Durability, 1438
Transport Address, 372
NDDS_Transport_Address_from_string, 375
NDDS_TRANSPORT_ADDRESS_INVALID, 377
NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER, 373
NDDS_Transport_Address_is_ipv4, 376
NDDS_Transport_Address_is_multicast, 377
NDDS_Transport_Address_print, 376
NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE, 374

NDDS_Transport_Address_to_string, 374
NDDS_Transport_Address_to_string_with_protocol_family_format, 375
Transport Plugins Configuration, 365
NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT, 368
NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC, 368
NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED, 367
NDDS_TRANSPORT_CLASSID_INVALID, 368
NDDS_TRANSPORT_CLASSID_RESERVED_RANGE, 370
NDDS_TRANSPORT_CLASSID_SHMEM, 369
NDDS_TRANSPORT_CLASSID_SHMEM_510, 369
NDDS_Transport_ClassId_t, 371
NDDS_TRANSPORT_CLASSID_TCPV4_LAN, 369
NDDS_TRANSPORT_CLASSID_TCPV4_WAN, 369
NDDS_TRANSPORT_CLASSID_TLSV4_LAN, 370
NDDS_TRANSPORT_CLASSID_TLSV4_WAN, 370
NDDS_TRANSPORT_CLASSID_UDPv4, 368
NDDS_TRANSPORT_CLASSID_UDPv4_WAN, 370
NDDS_TRANSPORT_CLASSID_UDPv6, 369
NDDS_TRANSPORT_CLASSID_UDPv6_510, 369
NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN, 370
NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN, 367
NDDS_TRANSPORT_LENGTH_UNLIMITED, 367
NDDS_TRANSPORT_PORT_INVALID, 367
NDDS_Transport_Port_t, 371
NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED, 370
NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MAX, 371
NDDS_TRANSPORT_UUID_SIZE, 367
NDDS_TRANSPORT_UUID_UNKNOWN, 367
TRANSPORT_BUILTIN, 548
transport_classid
NDDS_Transport_Interface_t, 1785
transport_info
dds::topic::ParticipantBuiltinTopicData, 1914
transport_info_list_max_length
rti::core::policy::DomainParticipantResourceLimits, 1428
TRANSPORT_MULTICAST, 549
TransportMulticastKind, 549
TransportMulticastSettingsSeq, 550
TRANSPORT_MULTICAST_MAPPING, 550
TRANSPORT_PRIORITY, 550
transport_priority
dds::topic::TopicBuiltinTopicData, 2497
transport_priority_mapping_high
NDDS_Transport_UDPv4_Property_t, 1805
NDDS_Transport_UDPv4_WAN_Property_t, 1814
NDDS_Transport_UDPv6_Property_t, 1825
transport_priority_mapping_low
NDDS_Transport_UDPv4_Property_t, 1805
NDDS_Transport_UDPv4_WAN_Property_t, 1814
NDDS_Transport_UDPv6_Property_t, 1825
transport_priority_mask
NDDS_Transport_UDPv4_Property_t, 1804
NDDS_Transport_UDPv4_WAN_Property_t, 1813
NDDS_Transport_UDPv6_Property_t, 1824
TRANSPORT_SELECTION, 551
TRANSPORT_UNICAST, 551
TransportUnicastSettingsSeq, 552
transport_uuid
NDDS_Transport_Property_t, 1793
TransportAllocationSettings_t, 2533
TransportBuiltin
rti::core::policy::TransportBuiltin, 2535
TransportBuiltinMask
rti::core::policy::TransportBuiltinMask, 2538, 2539
TransportClassId
rti::core, 706
TransportInfo
rti::core::TransportInfo, 2542
TransportInfoSeq
rti::core, 707
TransportMulticast
rti::core::policy::TransportMulticast, 2544, 2545
TransportMulticastKind
TRANSPORT_MULTICAST, 549
TransportMulticastSettings
rti::core::TransportMulticastSettings, 2550
TransportMulticastSettingsSeq
TRANSPORT_MULTICAST, 550
TransportPriority
dds::core::policy::TransportPriority, 2553
Transports, 75, 237
transports
rti::core::TransportMulticastSettings, 2550
rti::core::TransportUnicastSettings, 2561
rti::util::network_capture::NetworkCaptureParams, 1830
TransportSelection
rti::core::policy::TransportSelection, 2555
TransportUnicast
rti::core::policy::TransportUnicast, 2558
TransportUnicastSettings
rti::core::TransportUnicastSettings, 2560
TransportUnicastSettingsSeq
TRANSPORT_UNICAST, 552
trigger_flow
rti::pub::FlowController, 1575
trigger_value
dds::core::cond::Condition, 949

dds::core::cond::GuardCondition, 1595
trim_to_size
rti::core::xtypes::DynamicDataTypeSerializationProperty, 1502
truncate_journal
DURABILITY, 530
trust_algorithm_info
dds::topic::ParticipantBuiltinTopicData, 1913
dds::topic::PublicationBuiltinTopicData, 1989
dds::topic::SubscriptionBuiltinTopicData, 2438
trust_chain
rti::topic::trust::ParticipantTrustSignatureAlgorithmInfo, 1923
trust_protection_info
dds::topic::ParticipantBuiltinTopicData, 1913
dds::topic::PublicationBuiltinTopicData, 1989
dds::topic::SubscriptionBuiltinTopicData, 2437
TrustAlgorithmRequirements
rti::topic::trust::TrustAlgorithmRequirements, 2562
try_get
dds::core::policy::DataTag, 1136
rti::core::policy::Property, 1973
TSubscriptionBuiltinTopicData
dds::topic::SubscriptionBuiltinTopicData, 2430
type
dds::core::policy::DestinationOrderKind_def, 1264
dds::core::policy::DurabilityKind_def, 1442
dds::core::policy::HistoryKind_def, 1606
dds::core::policy::LivelinessKind_def, 1654
dds::core::policy::OwnershipKind_def, 1906
dds::core::policy::PresentationAccessScopeKind_def, 1951
dds::core::policy::ReliabilityKind_def, 2157
dds::core::policy::TypeConsistencyEnforcementKind_def, 2570
dds::core::QosProvider, 2044
dds::core::xtypes::DynamicData, 1482
dds::core::xtypes::ExtensibilityKind_def, 1551
dds::core::xtypes::Member, 1703
dds::core::xtypes::TypeKind_def, 2572
dds::core::xtypes::UnionMember, 2581
dds::topic::PublicationBuiltinTopicData, 1985
dds::topic::SubscriptionBuiltinTopicData, 2434
rti::config::activity_context::AttributeKindMask, 865
rti::config::LogCategory_def, 1682
rti::config::LogLevel_def, 1693
rti::config::PrintFormat_def, 1960
rti::config::Verbosity_def, 2613
rti::core::LocatorKind_def, 1681
rti::core::policy::AcknowledgmentKind_def, 797
rti::core::policy::CdrPaddingKind_def, 921
rti::core::policy::DataReaderInstanceRemovalKind_def, 1054
rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_d 1250
rti::core::policy::DestinationOrderScopeKind_def, 1265
rti::core::policy::IgnoredEntityReplacementKind_def, 1607
rti::core::policy::PublishModeKind_def, 2019
rti::core::policy::RemoteParticipantPurgeKind_def, 2164
rti::core::policy::ServiceKind_def, 2358
rti::core::policy::TransportMulticastKind_def, 2547
rti::core::policy::WireProtocolAutoKind_def, 2642
rti::core::ServiceRequestId_def, 2362
rti::core::ThreadSettingsCpuRotationKind_def, 2453
rti::core::TransportClassId_def, 2541
rti::pub::FlowControllerSchedulingPolicy_def, 1581
rti::sub::TopicQuerySelectionKind_def, 2529
rti::topic::PrintFormatKind_def, 1961
rti::util::heap_monitoring::SnapshotContentFormat_def, 2370
rti::util::heap_monitoring::SnapshotOutputFormat_def, 2371
Type System, 240
type_code_max_serialized_length
rti::core::policy::DomainParticipantResourceLimits, 1418
TYPE_CONSISTENCY_ENFORCEMENT, 552
TypeConsistencyEnforcementKind, 553
type_kind
dds::core::xtypes::DynamicData, 1483
type_lookup_service
rti::core::policy::DiscoveryConfigBuiltinChannelKindMask, 1321
type_lookup_service_reply
rti::core::policy::DiscoveryConfigBuiltinChannelKindMask, 1321
type_lookup_service_request
rti::core::policy::DiscoveryConfigBuiltinChannelKindMask, 1321
type_name
dds::pub::AnyDataWriter, 818
dds::sub::AnyDataReader, 810
dds::sub::DataReader< T >, 1023
dds::topic::AnyTopic, 827
dds::topic::PublicationBuiltinTopicData, 1980
dds::topic::SubscriptionBuiltinTopicData, 2430
dds::topic::TopicBuiltinTopicData, 2495
dds::topic::TopicDescription< T >, 2504
type_object_max_deserialized_length
rti::core::policy::DomainParticipantResourceLimits, 1420
type_object_max_serialized_length
rti::core::policy::DomainParticipantResourceLimits, 1419

TypeConsistencyEnforcement
dds::core::policy::TypeConsistencyEnforcement, 2565
TypeConsistencyEnforcementKind
TYPE_CONSISTENCY_ENFORCEMENT, 553
TypeKind
DynamicType and DynamicData, 440
TYPESUPPORT, 553
CdrPaddingKind, 554
TypeSupport
rti::core::policy::TypeSupport, 2575

UDP Transport Plugin definitions, 378
NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT, 379
NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT, 380
NDDS_Transport_UDP_Port, 380
NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT, 378
NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT, 380
NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT, 379
NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT, 379
NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX, 379
udp_debugging_address
NDDS_Transport_Shmem_Property_t, 1796
udp_debugging_port
NDDS_Transport_Shmem_Property_t, 1797
UDPv4
rti::core::LocatorKind_def, 1681
rti::core::policy::TransportBuiltin, 2535
rti::core::TransportClassId_def, 2541
udpv4
rti::core::policy::TransportBuiltinMask, 2539
UDPv4 Transport, 386
NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT, 390
NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS, 392
NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT, 393
NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER, 392
NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT, 390
NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT, 392
NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT, 392
NDDS_Transport_UDPv4_new, 393
NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX, 392
NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT, 390
NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT, 393
NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFA 391
NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFA 391
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAU 391
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX, 391
NDDS_Transport_UDPv4_string_to_address_cEA, 393
NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT, 390
UDPV4_ALIAS
rti::core::policy::TransportBuiltin, 2537
UDPV4_WAN
rti::core::LocatorKind_def, 1681
rti::core::policy::TransportBuiltin, 2536
rti::core::TransportClassId_def, 2541
udpv4_wan
rti::core::policy::TransportBuiltinMask, 2540
UDPV4_WAN_ALIAS
rti::core::policy::TransportBuiltin, 2537
UDPv6
rti::core::LocatorKind_def, 1681
rti::core::policy::TransportBuiltin, 2536
rti::core::TransportClassId_def, 2541
udpv6
rti::core::policy::TransportBuiltinMask, 2540
UDPv6 Transport, 399
NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT, 403
NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS, 405
NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER, 405
NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_M 404
NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT, 405
NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT, 405
NDDS_Transport_UDPv6_new, 406
NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX, 405
NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT, 403
NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFA 404

NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT, 404
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT, 404
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX, 404
NDDS_Transport_UDPv6_string_to_address_cEA, 405
UDPv6_510
rti::core::TransportClassId_def, 2541
UDPv6_ALIAS
rti::core::policy::TransportBuiltin, 2537
UINT_16_TYPE
dds::core::xtypes::TypeKind_def, 2572
UINT_32_TYPE
dds::core::xtypes::TypeKind_def, 2573
UINT_64_TYPE
dds::core::xtypes::TypeKind_def, 2573
UINT_8_TYPE
dds::core::xtypes::TypeKind_def, 2572
unacked_threshold
rti::queuing::ConsumerAvailabilityParams, 950
unacknowledged_sample_count
rti::core::status::ReliableWriterCacheChangedStatus, 2161
unacknowledged_sample_count_peak
rti::core::status::ReliableWriterCacheChangedStatus, 2161
uncommitted_sample_count
rti::core::status::DataReaderProtocolStatus, 1071
underlying
dds::core::safe_enum< def, inner >, 2268
UNICAST
rti::core::policy::TransportMulticastKind_def, 2547
unicast_enabled
NDDS_Transport_UDPv4_Property_t, 1801
NDDS_Transport_UDPv6_Property_t, 1821
unicast_locators
dds::topic::PublicationBuiltinTopicData, 1987
dds::topic::SubscriptionBuiltinTopicData, 2436
UNION_TYPE
dds::core::xtypes::TypeKind_def, 2574
UnionMember
dds::core::xtypes::UnionMember, 2580
UnionType
dds::core::xtypes::UnionType, 2585–2587
unit
dds::core::xtypes::AliasType, 802
UNKNOWN
rti::core::ServiceRequestId_def, 2362
unknown
dds::topic::BuiltinTopicKey, 906
rti::core::CoherentSetInfo, 938
rti::core::Guid, 1597
rti::core::ProductVersion, 1969
rti::core::SampleIdentity, 2284
rti::core::SequenceNumber, 2338
rti::core::VendorId, 2612
unload activation_library
Other Utilities, 598
unload_profiles
dds::core::QosProvider, 2045
unlock_condition
rti::core::cond::AsyncWaitSet, 852
unpack
dds::sub::SharedSamples< T, DELEGATE >, 2365, 2366
rti::sub, 769–771
unregister_contentfilter
dds::domain::DomainParticipant, 1350
unregister_instance
dds::pub::DataWriter< T >, 1160, 1161, 1173
unregister_thread
Supporting Types and Constants, 437
unregister_type
dds::domain::DomainParticipant, 1351
UNREGISTERED
rti::core::policy::DataWriterResourceLimitsInstanceReplacementKind_d 1251
unregistered_instance_count
rti::core::status::DataWriterCacheStatus, 1201
unregistered_instance_count_peak
rti::core::status::DataWriterCacheStatus, 1201
url_profile
rti::core::QosProviderParams, 2050, 2051
use_checksum
NDDS_Transport_UDPv4_Property_t, 1804
NDDS_Transport_UDPv4_WAN_Property_t, 1813
USE_DDS_DEFAULT_QOS_PROFILE
dds::core::QosProvider, 2047
use_standard_related_sample_pid
rti::config::compliance::RpcMask, 2220
user
Logging, 459
rti::config::LogCategory_def, 1683
rti::util::network_capture::ContentKindMask, 964
USER_DATA, 554
user_data
dds::topic::ParticipantBuiltinTopicData, 1910
dds::topic::PublicationBuiltinTopicData, 1982
dds::topic::SubscriptionBuiltinTopicData, 2432
user_endpoints_default_required_mask
rti::topic::trust::ParticipantTrustInterceptorAlgorithmInfo, 1919
user_forwarding_level
rti::core::MonitoringLoggingForwardingSettings, 1736
user_multicast

rti::core::policy::RtpsReservedPortKindMask, 2258
user_multicast_port_offset
rti::core::RtpsWellKnownPorts, 2264
user_unicast
rti::core::policy::RtpsReservedPortKindMask, 2258
user_unicast_port_offset
rti::core::RtpsWellKnownPorts, 2265
UserData
dds::core::policy::UserData, 2592, 2593
UseReaderContentFilter
rti::sub::TopicQuery, 2518
Utilities, 98

vacuum
rti::core::PersistentStorageSettings, 1933
valid
dds::sub::SampleInfo, 2289
valid_data
dds::sub::LoanedSamples< T >, 1671, 1672
rti::sub, 766, 767
valid_response_data
rti::pub::AcknowledgmentInfo, 796
ValidLoanedSamples
rti::sub::ValidLoanedSamples< T >, 2597
value
dds::core::KeyedBytesTopicType, 1627, 1628
dds::core::KeyedStringTopicType, 1630, 1631
dds::core::optional< T >, 1885, 1886
dds::core::policy::DataRepresentation, 1116
dds::core::policy::GroupData, 1593
dds::core::policy::OwnershipStrength, 1908
dds::core::policy::TopicData, 2502
dds::core::policy::TransportPriority, 2553, 2554
dds::core::policy::UserData, 2593, 2594
dds::core::xtypes::DynamicData, 1468–1470
dds::topic::BuiltinTopicKey, 906
rti::core::Cookie, 966
rti::core::optional_value< T >, 1895
rti::core::SequenceNumber, 2339, 2340
rti::core::VendorId, 2612
rti::sub::AckResponseData, 799
value_type
rti::core::bounded_sequence< T, MaxLength >, 891
rti::flat::SequenceIterator< E, OffsetKind >, 2331
rti::sub::ValidLoanedSamples< T >, 2596
vector
dds::core::vector< T >, 2605, 2606
vector_type
rti::core::bounded_sequence< T, MaxLength >, 891
vendor
rti::config::compliance::RpcMask, 2221
rti::config::compliance::XTypesMask, 2674
vendor_builtin_endpoints
dds::topic::ParticipantBuiltinTopicData, 1915
VendorId
rti::core::VendorId, 2612
VERBOSE
rti::config::PrintFormat_def, 1961
VERBOSE_TIMESTAMPED
rti::config::PrintFormat_def, 1961
Verbosity
Logging, 456
verbosity
rti::config::Logger, 1685, 1686
verbosity_by_category
rti::config::Logger, 1685, 1686
Version, 459
c_api_version, 461
c_build_number, 461
core_build_number, 461
core_version, 460
cpp_api_version, 461
cpp_build_number, 461
product_version, 461
request_reply_api_version, 460
request_reply_build_number, 460
view_state
dds::sub::status::DataState, 1124
rti::sub::status::DataStateEx, 1130, 1131
ViewState
dds::sub::status::ViewState, 2615
virtual_duplicate_dropped_sample_count
rti::core::status::DataReaderCacheStatus, 1050
virtual_guid
dds::topic::PublicationBuiltinTopicData, 1987
dds::topic::SubscriptionBuiltinTopicData, 2436
rti::core::policy::DataReaderProtocol, 1060, 1061
rti::core::policy::DataWriterProtocol, 1212
virtual_heartbeat_period
rti::core::policy::RtpsReliableWriterProtocol, 2237
VOLATILE
dds::core::policy::DurabilityKind_def, 1442
Volatile
dds::core::policy::Durability, 1438

wait
dds::core::cond::WaitSet, 2621–2623
rti::core::cond::AsyncWaitSetCompletionToken, 855
wait_for_acknowledgments
dds::pub::AnyDataWriter, 818
dds::pub::DataWriter< T >, 1168
dds::pub::Publisher, 2001
rti::queuing::QueueProducer< T >, 2086, 2087
rti::queuing::QueueReplier< TReq, TRep >, 2101
rti::queuing::QueueRequester< TReq, TRep >, 2121, 2122
wait_for_asynchronous_publishing
dds::pub::DataWriter< T >, 1174

dds::pub::Publisher, 2001
wait_for_historical_data
dds::sub::DataReader< T >, 1009
wait_for_replies
rti::queuing::QueueRequester< TReq, TRep >, 2123
rti::request::Requester< RequestType, ReplyType >, 2195, 2196
wait_for_requests
rti::queuing::QueueReplier< TReq, TRep >, 2103
rti::request::Replier< RequestType, ReplyType >, 2174
wait_for_samples
rti::queuing::QueueConsumer< T >, 2073
wait_for_service
dds::rpc::ClientEndpoint< Request, Reply >, 927
rti::request::Requester< RequestType, ReplyType >, 2190
wait_one
dds::core::cond::WaitSet, 2626
wait_timeout
rti::core::cond::AsyncWaitSetProperty, 860
WaitSet
dds::core::cond::WaitSet, 2621
Waitset Use Cases, 330
waitset_property
rti::core::cond::AsyncWaitSetProperty, 859
WaitSetImpl
dds::core::cond::WaitSet, 2621
WaitSetProperty
rti::core::cond::WaitSetProperty, 2628, 2629
wal_journal
DURABILITY, 530
walk_to
rpc_example::RobotControl, 2214
walk_to_async
rpc_example::RobotControlAsync, 2216
WARNING
rti::config::LogLevel_def, 1693
warning
Logging, 458
rti::config::Logger, 1690, 1691
rti::config::Verbosity_def, 2614
WEB_INTEGRATION
rti::core::policy::ServiceKind_def, 2358
WebIntegrationService
rti::core::policy::Service, 2353
what
dds::core::AlreadyClosedError, 806
dds::core::Error, 1540
dds::core::Exception, 1547
dds::core::IllegalOperationError, 1608
dds::core::ImmutablePolicyError, 1609
dds::core::InconsistentPolicyError, 1610
dds::core::InvalidArgumentError, 1619
dds::core::InvalidDowncastError, 1620
dds::core::NotAllowedBySecurityError, 1868
dds::core::NotEnabledError, 1869
dds::core::NullReferenceError, 1870
dds::core::OutOfResourcesError, 1899
dds::core::PreconditionNotMetError, 1942
dds::core::TimeoutError, 2473
dds::core::UnsupportedError, 2590
dds::rpc::RemoteInvalidArgumentError, 2162
dds::rpc::RemoteOutOfResourcesError, 2163
dds::rpc::RemoteUnknownExceptionError, 2166
dds::rpc::RemoteUnknownOperationError, 2167
dds::rpc::RemoteUnsupportedOperationError, 2168
rti::queuing::NoMatchingQueueException, 1836
WIRE_PROTOCOL, 555
all, 556
BackwardsCompatible, 558
default_mask, 557
Interoperable, 557
none, 556
WireProtocolAutoKind, 555
WireProtocol
rti::core::policy::WireProtocol, 2634
WireProtocolAutoKind
WIRE_PROTOCOL, 555
Working with IDL types, 606
write
dds::pub::DataWriter< T >, 1148, 1149, 1153–1157, 1179
write_noexcept
dds::pub::DataWriter< T >, 1180–1182
WriteParams
rti::pub::WriteParams, 2644
writer
rti::queuing::QueueProducer< T >, 2088
rti::queuing::QueueReplier< TReq, TRep >, 2106
rti::queuing::QueueRequester< TReq, TRep >, 2125
writer_attach
rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >, 2656
writer_compile
rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >, 2654
writer_compression_level
rti::core::CompressionSettings, 946
writer_compression_threshold
rti::core::CompressionSettings, 947
WRITER_DATA_LIFECYCLE, 554
writer_data_tag_list_max_length
rti::core::policy::DomainParticipantResourceLimits, 1431
writer_data_tag_string_max_length
rti::core::policy::DomainParticipantResourceLimits, 1431

writer_depth
dds::core::policy::Durability, 1439, 1440
writer_detach
rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >, 2656
writer_evaluate
rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >, 2655
writer_evaluate_helper
rti::topic::WriterContentFilterHelper< T, CompileData, WriterFilterData >, 2658
writer_finalize
rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >, 2655
writer_guid
rti::core::SampleIdentity, 2283
writer_instance_cache_allocation
rti::core::PersistentStorageSettings, 1934, 1935
writer_loaned_sample_allocation
rti::core::policy::DataWriterResourceLimits, 1247, 1248
writer_memory_state
rti::core::PersistentStorageSettings, 1936, 1937
writer_property_list_max_length
rti::core::policy::DomainParticipantResourceLimits, 1425
writer_property_string_max_length
rti::core::policy::DomainParticipantResourceLimits, 1426
writer_removed_batch_sample
rti::core::SampleFlag, 2281
writer_removed_batch_sample_dropped_sample_count
rti::core::status::DataReaderCacheStatus, 1050
writer_return_loan
rti::topic::WriterContentFilter< T, CompileData, WriterFilterData >, 2656
writer_sample_cache_allocation
rti::core::PersistentStorageSettings, 1935, 1936
writer_side_filter_optimization
rti::topic::ExpressionProperty, 1549
writer_user_data_max_length
rti::core::policy::DomainParticipantResourceLimits, 1416
WriterDataLifecycle
dds::core::policy::WriterDataLifecycle, 2661
wstring
Supporting Types and Constants, 435
WSTRING_TYPE
dds::core::xtypes::TypeKind_def, 2574
wstring_view
Supporting Types and Constants, 437
WStringType
dds::core::xtypes::WStringType, 2665
xcdr
DATA_REPRESENTATION, 520
xcdr2
DATA_REPRESENTATION, 521
XML
rti::topic::PrintFormatKind_def, 1962
Xml
rti::topic::PrintFormatProperty, 1966
xml
DATA_REPRESENTATION, 520
rti::core::xtypes, 720
XML Application Creation, 342
XTypesMask
rti::config::compliance::XTypesMask, 2668

ZERO
rti::core::policy::CdrPaddingKind_def, 921
zero
dds::core::Duration, 1450
dds::core::Time, 2459
rti::core::Guid, 1597
rti::core::SequenceNumber, 2338
Zero Copy Transfer Over Shared Memory, 50
zlib
rti::core::CompressionIdMask, 942