RTI TCP Transport

Generated by Doxygen 1.9.3

1 TCP Transport for RTI Connext **1**

1.1 Available Documentation . . . 1

1.2 Overview . . . 1

1.3 Configuring the TCP Transport . . . 2

2 Module Index **3**

2.1 Modules . . . 3

3 Data Structure Index **5**

3.1 Data Structures . . . 5

4 File Index **7**

4.1 File List . . . 7

5 Module Documentation **9**

5.1 Socket Utilities . . . 9

5.1.1 Detailed Description . . . 9

5.2 Connection Endpoint Abstractions . . . 9

5.3 TCP Transport Plugin API . . . 9

5.3.1 Detailed Description . . . 12

5.3.2 Macro Definition Documentation . . . 12

5.3.2.1 NDDS_TRANSPORT_CLASSNAME_TCPV4_LAN . . . 12

5.3.2.2 NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN . . . 12

5.3.2.3 NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER . . . 12

5.3.2.4 NDDS_TRANSPORT_TCPV4_MESSAGE_SIZE_MAX_DEFAULT . . . 12

5.3.2.5 NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT . . . 13

5.3.2.6 NDDS_TRANSPORT_TCPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT . . . 13

5.3.2.7 NDDS_TRANSPORT_TCPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT . . . 13

5.3.2.8 NDDS_TRANSPORT_TCPV4_SERVER_SOCKET_BACKLOG_DEFAULT . . . 13

5.3.2.9 NDDS_TRANSPORT_TCPV4_ADDRESS_BIT_COUNT . . . 13

5.3.2.10 NDDS_TRANSPORT_TCPV4_READ_BUFFER_POOL_GROWTH_POLICY_DEFAULT 13

5.3.2.11 NDDS_TRANSPORT_TCPV4_WRITE_BUFFER_POOL_GROWTH_POLICY_DEFAULT 14

5.3.2.12 NDDS_TRANSPORT_TCPV4_CONTROL_BUFFER_POOL_GROWTH_POLICY_DEFAULT 14

5.3.2.13 NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_FACTORY_GROWTH_POLICY_DEFAULT 14

5.3.2.14 NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_ATTRIBUTE_FACTORY_GROWTH_POLICY_DEFAUL

5.3.2.15 NDDS_TRANSPORT_TCPV4_CONNECTION_LIVELINESS_SETTINGS_DEFAULT . . 14

5.3.2.16 NDDS_TRANSPORT_TCPV4_EVENT_THREAD_SETTINGS_DEFAULT . . . 15

5.3.2.17 NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_DEFAULT . . . 15

5.3.2.18 NDDS_TRANSPORT_TCPV4_WINDOWS_IOCP_SETTINGS_DEFAULT . . . 15

5.3.2.19 NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT_LAN . . . 16

5.3.2.20 NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT . . . 17

5.3.2.21 NDDS_TRANSPORT_TCPV4_ADDRESS_LOCALHOST . . . 17
5.3.3 Typedef Documentation . . . 17
5.3.3.1 NDDS_Transport_TCPv4_OnConnectionEstablishedCallback . . . 17
5.3.3.2 NDDS_Transport_TCPv4_OnConnectionLostCallback . . . 18
5.3.4 Enumeration Type Documentation . . . 18
5.3.4.1 NDDS_Transport_TCPv4_OnConnectionLost_ReasonCode_t . . . 18
5.3.4.2 NDDS_Transport_TCPv4_SocketMonitoringKind_t . . . 19
5.3.5 Function Documentation . . . 19
5.3.5.1 NDDS_Transport_TCPv4_new() . . . 19
5.3.5.2 NDDS_Transport_TCPv4_create() . . . 20
5.3.5.3 NDDS_Transport_TCPv4_Plugin_stringToTransportAddress() . . . 20
5.4 Version . . . 21
5.4.1 Detailed Description . . . 21
5.4.2 Typedef Documentation . . . 21
5.4.2.1 NDDS_Transport_TCPv4_LibraryVersion . . . 21
5.4.3 Function Documentation . . . 22
5.4.3.1 NDDS_Transport_TCPv4_get_library_version() . . . 22
5.4.3.2 NDDS_Transport_TCPv4_get_build_version_string() . . . 22
5.5 TCP Transport Initial Peers . . . 22
5.6 Configure TCP Transport with Property QoS Policy . . . 22
5.7 TCP Transport Examples . . . 31
5.7.1 Detailed Description . . . 32
5.8 RTI TLS Support . . . 32
5.8.1 Detailed Description . . . 33
5.8.2 Macro Definition Documentation . . . 33
5.8.2.1 RTITLS_VERIFY_DEFAULT . . . 33
5.8.2.2 RTITLS_IDENTITY_DEFAULT . . . 33
5.8.2.3 RTITLS_CIPHER_LIST_DEFAULT . . . 34
5.8.2.4 RTITLS_CIPHER_LIST_ENCRYPT_HIGH . . . 34
5.8.2.5 RTITLS_CIPHER_LIST_UNENCRYPTED . . . 34
5.8.2.6 RTITLS_CIPHER_DEFAULT . . . 34
5.8.2.7 RTITLS_OPENSSL_CONFIGURATION_DEFAULT . . . 34
5.8.3 Typedef Documentation . . . 35
5.8.3.1 RTITLS_Verify_Callback . . . 35
5.8.4 Function Documentation . . . 35
5.8.4.1 RTITLS_thread_exit() . . . 35
5.8.4.2 RTITLS_default_verify_callback() . . . 35
5.8.4.3 RTITLS_verbose_verify_callback() . . . 35
5.9 C Example of TCP Transport with PropertiesQosPolicy Configuration . . . 36

5.9.1 Detailed Description . . . 36
5.10 HelloWorld.idl . . . 39
5.11 HelloCommon.h . . . 40
5.12 HelloWorld.c . . . 46
5.13 HelloWorld_publisher.c . . . 51
5.14 HelloWorld_subscriber.c . . . 54
5.15 HelloWorldPlugin.c . . . 58
5.16 HelloWorldSupport.c . . . 69

6 Data Structure Documentation **73**

6.1 NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t Struct Reference . . . 73
6.1.1 Detailed Description . . . 73
6.1.2 Field Documentation . . . 73
6.1.2.1 enable . . . 73
6.1.2.2 lease_duration . . . 74
6.1.2.3 assertions_per_lease_duration . . . 74
6.2 NDDS_Transport_TCPv4_EventThreadSettings_t Struct Reference . . . 74
6.2.1 Detailed Description . . . 74
6.2.2 Field Documentation . . . 74
6.2.2.1 priority . . . 74
6.2.2.2 stack_size . . . 75
6.2.2.3 options . . . 75
6.3 NDDS_Transport_TCPv4_Property_t Struct Reference . . . 75
6.3.1 Detailed Description . . . 77
6.3.2 Field Documentation . . . 77
6.3.2.1 parent . . . 78
6.3.2.2 send_socket_buffer_size . . . 78
6.3.2.3 recv_socket_buffer_size . . . 78
6.3.2.4 ignore_loopback_interface . . . 78
6.3.2.5 ignore_nonrunning_interfaces . . . 79
6.3.2.6 transport_priority_mask . . . 79
6.3.2.7 transport_priority_mapping_low . . . 80
6.3.2.8 transport_priority_mapping_high . . . 80
6.3.2.9 force_interface_poll_detection . . . 80
6.3.2.10 interface_poll_period . . . 81
6.3.2.11 server_socket_backlog . . . 81
6.3.2.12 public_address . . . 81
6.3.2.13 bind_interface_address . . . 81
6.3.2.14 server_bind_port . . . 82

6.3.2.15 read_buffer_allocation . . . 82
6.3.2.16 write_buffer_allocation . . . 82
6.3.2.17 shared_write_buffer_allocation . . . 83
6.3.2.18 control_buffer_allocation . . . 83
6.3.2.19 control_message_allocation . . . 83
6.3.2.20 control_attribute_allocation . . . 83
6.3.2.21 force_asynchronous_send . . . 84
6.3.2.22 on_connection_established . . . 84
6.3.2.23 on_connection_lost . . . 84
6.3.2.24 max_packet_size . . . 85
6.3.2.25 enable_keep_alive . . . 85
6.3.2.26 keep_alive_time . . . 85
6.3.2.27 keep_alive_interval . . . 85
6.3.2.28 keep_alive_retry_count . . . 86
6.3.2.29 user_timeout . . . 86
6.3.2.30 connection_liveliness_settings . . . 86
6.3.2.31 event_thread_settings . . . 86
6.3.2.32 disable_nagle . . . 86
6.3.2.33 logging_verbosity_bitmap . . . 87
6.3.2.34 security_logging_verbosity_bitmap . . . 87
6.3.2.35 socket_monitoring_kind . . . 88
6.3.2.36 windows_iocp_settings . . . 88
6.3.2.37 send_crc . . . 88
6.3.2.38 force_crc_check . . . 88
6.3.2.39 outstanding_connection_cookies . . . 88
6.3.2.40 outstanding_connection_cookies_life_span . . . 89
6.3.2.41 send_max_wait_sec . . . 89
6.3.2.42 client_connection_negotiation_timeout . . . 89
6.3.2.43 server_connection_negotiation_timeout . . . 90
6.3.2.44 initial_handshake_timeout . . . 90
6.3.2.45 tls . . . 90
6.3.2.46 disable_interface_tracking . . . 91
6.4 NDDS_Transport_TCPv4_WindowsIOCPSettings_t Struct Reference . . . 91
6.4.1 Detailed Description . . . 91
6.4.2 Field Documentation . . . 91
6.4.2.1 concurrency_value . . . 91
6.4.2.2 thread_pool_size . . . 91
6.5 RTITLS_Ciphers Struct Reference . . . 92
6.5.1 Detailed Description . . . 92

6.5.2 Field Documentation . . . 92
6.5.2.1 cipher_list . . . 92
6.5.2.2 ciphersuites . . . 93
6.5.2.3 dh_param_files_length . . . 93
6.5.2.4 dh_param_files . . . 93
6.5.2.5 engine_id . . . 93
6.6 RTITLS_DHParamFile Struct Reference . . . 93
6.6.1 Detailed Description . . . 94
6.6.2 Field Documentation . . . 94
6.6.2.1 file . . . 94
6.6.2.2 bits . . . 94
6.7 RTITLS_Identity Struct Reference . . . 94
6.7.1 Detailed Description . . . 95
6.7.2 Field Documentation . . . 95
6.7.2.1 certificate_chain . . . 95
6.7.2.2 certificate_chain_file . . . 95
6.7.2.3 private_key_password . . . 96
6.7.2.4 private_key . . . 96
6.7.2.5 private_key_file . . . 96
6.7.2.6 rsa_private_key . . . 96
6.7.2.7 rsa_private_key_file . . . 96
6.8 RTITLS_OpenSSL_Configuration Struct Reference . . . 97
6.8.1 Detailed Description . . . 97
6.8.2 Field Documentation . . . 97
6.8.2.1 verify . . . 97
6.8.2.2 identity . . . 97
6.8.2.3 cipher . . . 98
6.9 RTITLS_Verification Struct Reference . . . 98
6.9.1 Detailed Description . . . 98
6.9.2 Field Documentation . . . 98
6.9.2.1 ca_file . . . 98
6.9.2.2 ca_path . . . 99
6.9.2.3 ca . . . 99
6.9.2.4 verify_depth . . . 99
6.9.2.5 callback . . . 99
6.9.2.6 crl_file . . . 99

7 File Documentation **101**

7.1 rtitls_openssl.h . . . 101

Index **107**

Chapter 1

TCP Transport for RTI Connext

Real-Time Innovations, Inc.

The RTI TCP Transport enables communication between RTI Connext applications using TCP. The communication can be either over a LAN or over a WAN.

1.1 Available Documentation

This document contains:

- **Overview** (p. 1)
- **Configuring the TCP Transport** (p. 2)

For additional information, please see the following PDF document:

- **RTI Connext DDS Core Libraries User's Manual**

1.2 Overview

Out of the box, RTI Connext uses UDP and Shared Memory to communicate with other DDS applications. This configuration is appropriate for systems running within a single LAN. However, using the UDP transport introduces some problems when DDS applications in different LANs need to communicate:

- UDP traffic is usually filtered out by the LAN firewalls for security reasons.
- Forwarded ports are usually TCP ports.
- Each LAN may run in its own private IP address space and use NAT (Network Address Translation) to communicate with other networks.

RTI TCP Transport enables participant discovery and data exchange using the TCP protocol either on a LAN, or over the public WAN. RTI TCP Transport enables RTI Connext to address the challenges of using TCP as a low-level communication mechanism between peers and limits the number of exposed ports to one. When using the default UDP transport, an RTI Connext application uses multiple UDP ports for communication, which may make it unsuitable for deployment across firewalled networks.

1.3 Configuring the TCP Transport

The TCP transport is distributed as a shared library located under <RTI Connext installation directory>/lib/<architecture>. The library is called **nddstransporttcp**.

There are two ways in which this transport can be configured:

1. By setting up predefined property strings in the Property QoS Policy of the DomainParticipant (on architectures that support shared libraries).

   Refer to **Configure TCP Transport with Property QoS Policy** (p. 22) for the predefined property strings.

   With this first approach, RTI Connext will dynamically load the TCP transport library at run time and then implicitly create and register the transport plugin.

   (Shared 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.)

2. By instantiating a new transport and registering it with the DomainParticipant (available in C/C++ API only).

   Refer to **NDDS_Transport_TCPv4_new** (p. 19) for information on how to create a new TCP transport instance.

   Refer to **NDDS_Transport_Support_register_transport** for documentation on how to register the transport instance with RTI Connext.

   To use this second approach, you need access to the **TCP Transport Plugin API** (p. 9) at compile time. Your application will have to include the following header file:

   ```
   #include "ndds/transport_tcp/transport_tcp_tcpv4.h"
   ```

   (The above path assumes that RTI TCP Transport is installed in the same directory as RTI Connext.)

Regardless of which way is used to configure the TCP transport, you will need to specify TCP as the transport in the initial peers. See **TCP Transport Initial Peers** (p. 22) for details.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

- Socket Utilities . . . 9
  - Connection Endpoint Abstractions . . . 9
- TCP Transport Plugin API . . . 9
- Version . . . 21
- TCP Transport Initial Peers . . . 22
- Configure TCP Transport with Property QoS Policy . . . 22
- TCP Transport Examples . . . 31
  - C Example of TCP Transport with PropertiesQosPolicy Configuration . . . 36
    - HelloWorld.idl . . . 39
    - HelloCommon.h . . . 40
    - HelloWorld.c . . . 46
    - HelloWorld_publisher.c . . . 51
    - HelloWorld_subscriber.c . . . 54
    - HelloWorldPlugin.c . . . 58
    - HelloWorldSupport.c . . . 69
- RTI TLS Support . . . 32

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t
Connection liveliness settings . . . 73
NDDS_Transport_TCPv4_EventThreadSettings_t
Event thread settings . . . 74
NDDS_Transport_TCPv4_Property_t
Configurable TCP Transport-Plugin properties . . . 75
NDDS_Transport_TCPv4_WindowsIOCPSettings_t
Windows I/O completion ports settings . . . 91
RTITLS_Ciphers
Set of TLS properties for cipher configuration . . . 92
RTITLS_DHParamFile
Name of a Diffie-Helman (DH) key file and the length of the contained key in bits . . . 93
RTITLS_Identity
Set of TLS properties for identity . . . 94
RTITLS_OpenSSL_Configuration
Full set of TLS properties . . . 97
RTITLS_Verification
Set of TLS properties for certificate authorities (CAs) and verification . . . 98

Chapter 4

File Index

4.1 File List

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

rtitls_openssl.h 101

Chapter 5

Module Documentation

5.1 Socket Utilities

Socket Utilities.

Modules

- **Connection Endpoint Abstractions**
  Connection Endpoint and Factory Abstractions.

5.1.1 Detailed Description

Socket Utilities.

5.2 Connection Endpoint Abstractions

Connection Endpoint and Factory Abstractions.

Connection Endpoint and Factory Abstractions.

5.3 TCP Transport Plugin API

TCP Transport Plugin interfaces and definitions.

Data Structures

- struct **NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t**

  Connection liveliness settings.
- struct **NDDS_Transport_TCPv4_EventThreadSettings_t**

  Event thread settings.
- struct **NDDS_Transport_TCPv4_WindowsIOCPSettings_t**

  Windows I/O completion ports settings.
- struct **NDDS_Transport_TCPv4_Property_t**

  Configurable TCP Transport-Plugin properties.

Macros

- #define **NDDS_TRANSPORT_CLASSNAME_TCPV4_LAN** "tcpv4_lan"

  IPv4 TCP/IP Transport-Plugin class name for LAN case.
- #define **NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN** "tcpv4_wan"

  IPv4 TCP/IP Transport-Plugin class name for WAN case.
- #define **NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER** 7400

  The default value for the server bind port.
- #define **NDDS_TRANSPORT_TCPV4_MESSAGE_SIZE_MAX_DEFAULT** (65536)

  Default value of maximum message size.
- #define **NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT** (-1)

  Used to specify that os default should be used for the socket buffer size.
- #define **NDDS_TRANSPORT_TCPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT** (131072)

  Default value of NDDS_Transport_Property_t::send_socket_buffer_size.
- #define **NDDS_TRANSPORT_TCPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT** (131072)

  Default value of NDDS_Transport_Property_t::recv_socket_buffer_size.
- #define **NDDS_TRANSPORT_TCPV4_SERVER_SOCKET_BACKLOG_DEFAULT** (5)

  The max number of pending connections waiting for the control thread to accept them.
- #define **NDDS_TRANSPORT_TCPV4_ADDRESS_BIT_COUNT** (64)

  Number of signficant bits of 16-byte address.
- #define **NDDS_TRANSPORT_TCPV4_READ_BUFFER_POOL_GROWTH_POLICY_DEFAULT NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT**

  The default settings for the buffer pool used to provide memory for the received messages.
- #define **NDDS_TRANSPORT_TCPV4_WRITE_BUFFER_POOL_GROWTH_POLICY_DEFAULT** {4L, 32L, 2L}

  The default settings for the buffer used to provide memory for the bytes being written using asynchronous send.
- #define **NDDS_TRANSPORT_TCPV4_CONTROL_BUFFER_POOL_GROWTH_POLICY_DEFAULT NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT**

  The default settings for the buffer pool used to provide memory for the control messages read or written.
- #define **NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_FACTORY_GROWTH_POLICY_DEFAULT NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT**

  The default settings for the buffer used to provide memory for the control message factory.
- #define **NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_ATTRIBUTE_FACTORY_GROWTH_POLICY_DEFAULT NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT**

  The default settings for the buffer used to provide memory for the control message attribute factory.
- #define **NDDS_TRANSPORT_TCPV4_CONNECTION_LIVELINESS_SETTINGS_DEFAULT**

  *Use this to initialize a **NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t** (p. 73) structure.*

- #define **NDDS_TRANSPORT_TCPV4_EVENT_THREAD_SETTINGS_DEFAULT**

  *Use this to initialize a **NDDS_Transport_TCPv4_EventThreadSettings_t** (p. 74) structure.*
- #define **NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_DEFAULT** **NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_SELECT**

  Default value for the socket monitoring API.
- #define **NDDS_TRANSPORT_TCPV4_WINDOWS_IOCP_SETTINGS_DEFAULT**

  *Use this to initialize a **NDDS_Transport_TCPv4_WindowsIOCPSettings_t** (p. 91) structure.*
- #define **NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT_LAN**

  *Use this to initialize a **NDDS_Transport_TCPv4_Property_t** (p. 75) structure for LAN communication.*
- #define **NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT** **NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT_LAN**

  *Use this to initialize a **NDDS_Transport_TCPv4_Property_t** (p. 75) structure.*
- #define **NDDS_TRANSPORT_TCPV4_ADDRESS_LOCALHOST** {{0,0,0,0, 0,0,0,0, 0xff,0xff, 0x1c, 0xe8, 127,0,0,1}}

  The definition of localhost for the TCP transport: 127.0.0.1:7400.

Typedefs

- typedef void(∗ **NDDS_Transport_TCPv4_OnConnectionEstablishedCallback**) (RTI_UINT32 remote_peer_addresss, RTI_UINT16 remote_peer_port)

  Prototype of the function that can be called when a new connection is successfully established with a remote peer.
- typedef void(∗ **NDDS_Transport_TCPv4_OnConnectionLostCallback**) (RTI_UINT32 remote_peer_addresss, RTI_UINT16 remote_peer_port, **NDDS_Transport_TCPv4_OnConnectionLost_ReasonCode_t** reason)

  Prototype of the function that can be called when an existing connection is lost.

Enumerations

- enum **NDDS_Transport_TCPv4_OnConnectionLost_ReasonCode_t** {
  NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_UNKNOWN = 0 ,
  NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_CODE_FINALIZED ,
  NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_CODE_BROKEN_PIPE ,
  NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_DESTROYED }

  A set of possible reasons why a connection is lost.
- enum **NDDS_Transport_TCPv4_SocketMonitoringKind_t** {
  NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_WINDOWS_WAITFORMULTIPLEOBJECTS = 0 ,
  NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_SELECT = 1 ,
  NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_WINDOWS_IOCP = 2 ,
  NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_POLL = 3 }

  Possible values for the socket monitoring API.

Functions

- NDDS_Transport_Plugin ∗ **NDDS_Transport_TCPv4_new** (const struct **NDDS_Transport_TCPv4_Property_t** ∗property_in)

  Creates an instance of the TCP transport plugin and initializes it from the specified TCP transport property structure.
- NDDS_Transport_Plugin ∗ **NDDS_Transport_TCPv4_create** (**NDDS_Transport_Address_t** ∗default_network_address_out, const struct **DDS_PropertyQosPolicy** ∗property_in)

  Creates an instance of a TCP Transport Plugin, using PropertyQosPolicy.
- RTIBool **NDDS_Transport_TCPv4_Plugin_stringToTransportAddress** (**NDDS_Transport_Address_t** ∗address_out, const char ∗address_in, RTIBool acceptHostName_in)

  Converts a string into a transport address suitable to be used with the TCP transport.

5.3.1 Detailed Description

TCP Transport Plugin interfaces and definitions.

5.3.2 Macro Definition Documentation

5.3.2.1 NDDS_TRANSPORT_CLASSNAME_TCPV4_LAN

```
#define NDDS_TRANSPORT_CLASSNAME_TCPV4_LAN "tcpv4_lan"
```

IPv4 TCP/IP Transport-Plugin class name for LAN case.

5.3.2.2 NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN

```
#define NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN "tcpv4_wan"
```

IPv4 TCP/IP Transport-Plugin class name for WAN case.

5.3.2.3 NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER

```
#define NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER 7400
```

The default value for the server bind port.

5.3.2.4 NDDS_TRANSPORT_TCPV4_MESSAGE_SIZE_MAX_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_MESSAGE_SIZE_MAX_DEFAULT (65536)
```

Default value of maximum message size.

5.3.2.5 NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT (-1)
```

Used to specify that os default should be used for the socket buffer size.

5.3.2.6 NDDS_TRANSPORT_TCPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT (131072)
```

Default value of NDDS_Transport_Property_t::send_socket_buffer_size.

5.3.2.7 NDDS_TRANSPORT_TCPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT (131072)
```

Default value of NDDS_Transport_Property_t::recv_socket_buffer_size.

5.3.2.8 NDDS_TRANSPORT_TCPV4_SERVER_SOCKET_BACKLOG_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_SERVER_SOCKET_BACKLOG_DEFAULT (5)
```

The max number of pending connections waiting for the control thread to accept them.

5.3.2.9 NDDS_TRANSPORT_TCPV4_ADDRESS_BIT_COUNT

```
#define NDDS_TRANSPORT_TCPV4_ADDRESS_BIT_COUNT (64)
```

Number of signficant bits of 16-byte address.

5.3.2.10 NDDS_TRANSPORT_TCPV4_READ_BUFFER_POOL_GROWTH_POLICY_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_READ_BUFFER_POOL_GROWTH_POLICY_DEFAULT NDDS_TRANSPORT_ALLOCATION_↩
SETTINGS_DEFAULT
```

The default settings for the buffer pool used to provide memory for the received messages.

5.3.2.11 NDDS_TRANSPORT_TCPV4_WRITE_BUFFER_POOL_GROWTH_POLICY_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_WRITE_BUFFER_POOL_GROWTH_POLICY_DEFAULT {4L, 32L, 2L}
```

The default settings for the buffer used to provide memory for the bytes being written using asynchronous send.

5.3.2.12 NDDS_TRANSPORT_TCPV4_CONTROL_BUFFER_POOL_GROWTH_POLICY_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_CONTROL_BUFFER_POOL_GROWTH_POLICY_DEFAULT NDDS_TRANSPORT_ALLOCATION↩
_SETTINGS_DEFAULT
```

The default settings for the buffer pool used to provide memory for the control messages read or written.

5.3.2.13 NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_FACTORY_GROWTH_POLICY_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_FACTORY_GROWTH_POLICY_DEFAULT NDDS_TRANSPORT_↩
ALLOCATION_SETTINGS_DEFAULT
```

The default settings for the buffer used to provide memory for the control message factory.

5.3.2.14 NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_ATTRIBUTE_FACTORY_GROWTH_POLICY_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_ATTRIBUTE_FACTORY_GROWTH_POLICY_DEFAULT NDDS_↩
TRANSPORT_ALLOCATION_SETTINGS_DEFAULT
```

The default settings for the buffer used to provide memory for the control message attribute factory.

5.3.2.15 NDDS_TRANSPORT_TCPV4_CONNECTION_LIVELINESS_SETTINGS_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_CONNECTION_LIVELINESS_SETTINGS_DEFAULT
```

Value:

```
{ \
0,  /* enable */ \
10, /* lease_duration */ \
3   /* assertions_per_lease_duration */ \
}
```

Use this to initialize a **NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t** (p. 73) structure.

5.3.2.16 NDDS_TRANSPORT_TCPV4_EVENT_THREAD_SETTINGS_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_EVENT_THREAD_SETTINGS_DEFAULT
```

Value:

```
        { \
        RTI_OSAPI_THREAD_PRIORITY_DEFAULT, /* priority */ \
        RTI_OSAPI_THREAD_STACK_SIZE_DEFAULT, /* stack_size */ \
        RTI_OSAPI_THREAD_OPTION_PRIORITY_ENFORCE | RTI_OSAPI_THREAD_OPTION_STDIO /* options */ \
}
```

Use this to initialize a **NDDS_Transport_TCPv4_EventThreadSettings_t** (p. 74) structure.

5.3.2.17 NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_DEFAULT   NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING↩
_KIND_SELECT
```

Default value for the socket monitoring API.

NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_WINDOWS_IOCP on Windows systems, NDDS_↩ TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_SELECT on the other systems.

5.3.2.18 NDDS_TRANSPORT_TCPV4_WINDOWS_IOCP_SETTINGS_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_WINDOWS_IOCP_SETTINGS_DEFAULT
```

Value:

```
 { \
 1, /* windows_iocp_concurrency_value */ \
 2, /* windows_iocp_thread_pool_size */ \
 RTI_OSAPI_THREAD_PRIORITY_DEFAULT,   /* windows_iocp_thread_priority */ \
 RTI_OSAPI_THREAD_STACK_SIZE_DEFAULT, /* windows_iocp_thread_stack_size */ \
}
```

Use this to initialize a **NDDS_Transport_TCPv4_WindowsIOCPSettings_t** (p. 91) structure.

5.3.2.19 NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT_LAN

```
#define NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT_LAN
```

Value:

```
  { \
    { /* parent */ \
      NDDS_TRANSPORT_CLASSID_TCPV4_LAN, /* classid */ \
      NDDS_TRANSPORT_TCPV4_ADDRESS_BIT_COUNT, /* address_bit_count */ \
      NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED, \
              /* properties_bitmap */ \
      128, /* gather_send_buffer_count_max */ \
      NDDS_TRANSPORT_TCPV4_MESSAGE_SIZE_MAX_DEFAULT, /* message_size_max */ \
      NULL, /* allow_interfaces_list */ \
      0,    /* allow_interfaces_list_length */ \
      NULL, /* deny_interfaces_list */ \
      0,    /* deny_interfaces_list_length;*/ \
      NULL, /* allow_multicast_interfaces_list */ \
      0,    /* allow_multicast_interfaces_list_length */ \
      NULL, /* deny_multicast_interfaces_list */ \
      0,    /* deny_multicast_interfaces_list_length */ \
      NDDS_TRANSPORT_UUID_UNKNOWN,     /* transport_uuid */ \
      NULL,                            /* domain_participant_ptr */ \
      NULL,                            /* domain_participant_forwarder */ \
      NULL,                            /* thread_name_prefix */ \
      NDDS_TRANSPORT_LENGTH_UNLIMITED, /* max_interface_count */ \
    }, \
    NDDS_TRANSPORT_TCPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT, \
            /* send_socket_buffer_size */ \
    NDDS_TRANSPORT_TCPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT, \
            /* recv_socket_buffer_size */ \
    1, /* ignore_loopback_interface */ \
    1, /* ignore_nonrunning_interfaces */ \
    0, \
    0, \
    0xffL, /* transport priority mapping */ \
    0,     /* force_interface_poll_detection */ \
    500,   /* interface_poll_period (in millisec) */ \
    NDDS_TRANSPORT_TCPV4_TRANSPORT_MODE_SYMMETRIC, \
            /* transport_mode (deprecated) */ \
    NDDS_TRANSPORT_TCPV4_SERVER_SOCKET_BACKLOG_DEFAULT, \
            /* server_socket_backlog */ \
    NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER, /* public_address */ \
    NULL,                                       /* bind_interface_address */ \
    NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER,   /*server_bind_port */ \
    NDDS_TRANSPORT_TCPV4_READ_BUFFER_POOL_GROWTH_POLICY_DEFAULT, \
            /* read_buffer_allocation */ \
    NDDS_TRANSPORT_TCPV4_WRITE_BUFFER_POOL_GROWTH_POLICY_DEFAULT, \
            /* write_buffer_allocation */ \
    0, /* shared_write_buffer_allocation */ \
    1, /* enable_write_buffer_replacement */ \
    NDDS_TRANSPORT_TCPV4_CONTROL_BUFFER_POOL_GROWTH_POLICY_DEFAULT, \
            /* control_buffer_allocation */ \
    NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_FACTORY_GROWTH_POLICY_DEFAULT, \
            /* control_message_allocation */ \
    NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_ATTRIBUTE_FACTORY_GROWTH_POLICY_DEFAULT, \
            /* control_attribute_allocation */ \
    0,    /* force_asynchronous_send */ \
    NULL, /* on_connection_established */ \
    NULL, /* on_connection_lost */ \
    -1L,  /* max_packet_size */ \
    0,    /* enable_keep_alive */ \
    -1L,  /* keep_alive_time */ \
    -1L,  /* keep_alive_internal */ \
    -1L,  /* keep_alive_retry_count */ \
    0,    /* user_timeout */ \
    NDDS_TRANSPORT_TCPV4_CONNECTION_LIVELINESS_SETTINGS_DEFAULT, \
            /* connection_liveliness_settings */ \
    NDDS_TRANSPORT_TCPV4_EVENT_THREAD_SETTINGS_DEFAULT, \
            /* event_thread_settings */ \
    0,   /* disable_nagle */ \
    -1L, /* logging_verbosity_bitmap */ \
    -1L, /* security_logging_verbosity_bitmap */ \
    -1L, /* use_windows_select */ \
    NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_DEFAULT, \
            /* socket_monitoring_kind */ \
    NDDS_TRANSPORT_TCPV4_WINDOWS_IOCP_SETTINGS_DEFAULT, \
            /* windows_iocp_settings */ \
```

```
    0,   /*send_crc*/ \
    0,   /*force_crc_check*/ \
    0,   /* negotiate_session_id */ \
    100, /* outstanding_connection_cookies */ \
    -1L, /* outstanding_connection_cookies_life_span */ \
    3,   /* send_max_wait_sec */ \
    10,  /* client_connection_negotiation_timeout */ \
    10,  /* server_connection_negotiation_timeout */ \
    10,  /* initial_handshake_timeout */ \
    RTITLS_OPENSSL_CONFIGURATION_DEFAULT, /* tls */ \
    NULL, /* connection_endpoint_factory_create_fcn */ \
    NULL, /* connection_endpoint_factory_delete_fcn */ \
    0,    /* disable_interface_tracking */ \
    NULL  /* reserved */ \
  }
```

Use this to initialize a **NDDS_Transport_TCPv4_Property_t** (p. 75) structure for LAN communication.

5.3.2.20 NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT

```
#define NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT   NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT_LAN
```

Use this to initialize a **NDDS_Transport_TCPv4_Property_t** (p. 75) structure.

5.3.2.21 NDDS_TRANSPORT_TCPV4_ADDRESS_LOCALHOST

```
#define NDDS_TRANSPORT_TCPV4_ADDRESS_LOCALHOST {{0,0,0,0, 0,0,0,0, 0xff,0xff, 0x1c, 0xe8, 127,0,0,1}}
```

The definition of localhost for the TCP transport: 127.0.0.1:7400.

5.3.3 Typedef Documentation

5.3.3.1 NDDS_Transport_TCPv4_OnConnectionEstablishedCallback

```
typedef void(* NDDS_Transport_TCPv4_OnConnectionEstablishedCallback) (RTI_UINT32 remote_peer_↩
addresss, RTI_UINT16 remote_peer_port)
```

Prototype of the function that can be called when a new connection is successfully established with a remote peer.

Parameters

remote_peer_addresss	the address of the remote peer connected encoded as network-ordered IPv4 address.
remote_peer_port	the port number of the remote connection encoded as network-ordered 16-bit value

5.3.3.2 NDDS_Transport_TCPv4_OnConnectionLostCallback

```
typedef void(* NDDS_Transport_TCPv4_OnConnectionLostCallback) (RTI_UINT32 remote_peer_addresss,
RTI_UINT16 remote_peer_port,  NDDS_Transport_TCPv4_OnConnectionLost_ReasonCode_t reason)
```

Prototype of the function that can be called when an existing connection is lost.

This method is always called after the on_connection_established is called.

It is also called if a sendresource or recvresource is destroyed (after closing the existing connections), and on shutdown

Parameters

remote_peer_addresss	the address of the remote peer connected encoded as network-ordered IPv4 address.
remote_peer_port	the port number of the remote connection encoded as network-ordered 16-bit value
reason	an enumeration that describes the reason of the disconnection

5.3.4 Enumeration Type Documentation

5.3.4.1 NDDS_Transport_TCPv4_OnConnectionLost_ReasonCode_t

```
enum  NDDS_Transport_TCPv4_OnConnectionLost_ReasonCode_t
```

A set of possible reasons why a connection is lost.

See also

NDDS_Transport_TCPv4_OnConnectionLostCallback (p. 18)

Enumerator

NDDS_TRANSPORT_TCPV4_ON_CONNECTION_↩ LOST_REASON_UNKNOWN	The reason why the connection was closed is not known
NDDS_TRANSPORT_TCPV4_ON_CONNECTION_↩ LOST_REASON_CODE_FINALIZED	Server side: connection closed by remote client. Client side: the internal send resource was destroyed.
NDDS_TRANSPORT_TCPV4_ON_CONNECTION_↩ LOST_REASON_CODE_BROKEN_PIPE	Describes a condition where the remote peer has ungracefully closed the connection (app was crashed or suddenly interrupted)
NDDS_TRANSPORT_TCPV4_ON_CONNECTION_↩ LOST_REASON_DESTROYED	Describes a condition where the current node has decided to close the connection.

5.3.4.2 NDDS_Transport_TCPv4_SocketMonitoringKind_t

```
enum NDDS_Transport_TCPv4_SocketMonitoringKind_t
```

Possible values for the socket monitoring API.

The socket monitoring API is configured using the property socket_monitoring_kind

See also

NDDS_Transport_TCPv4_Property_t (p. 75).

Enumerator

NDDS_TRANSPORT_TCPV4_SOCKET_↩ MONITORING_KIND_WINDOWS_↩ WAITFORMULTIPLEOBJECTS	The transport uses the API WaitForMultipleObjects to monitor sockets. This value only applies to Windows systems.
NDDS_TRANSPORT_TCPV4_SOCKET_↩ MONITORING_KIND_SELECT	The transport uses the POSIX select API to monitor sockets.
NDDS_TRANSPORT_TCPV4_SOCKET_↩ MONITORING_KIND_WINDOWS_IOCP	The transport uses Windows I/O completion ports to monitor sockets. This value only applies to Windows systems.
NDDS_TRANSPORT_TCPV4_SOCKET_↩ MONITORING_KIND_POLL	The transport uses POSIX poll to monitor sockets. This value currently only supported in linux systems.

5.3.5 Function Documentation

5.3.5.1 NDDS_Transport_TCPv4_new()

```
NDDS_Transport_Plugin * NDDS_Transport_TCPv4_new (
            const struct  NDDS_Transport_TCPv4_Property_t * property_in )
```

Creates an instance of the TCP transport plugin and initializes it from the specified TCP transport property structure.

This is the main initializer of the TCP transport plugin. To use the default settings, pass a pointer to a struct **NDDS_↩ Transport_TCPv4_Property_t** (p. 75) initialized with the macro **NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT** (p. 17).

Parameters

property_in	<<***in***>> A pointer to a struct **NDDS_Transport_TCPv4_Property_t** (p. 75) containing the transport property. The plugin will make a copy of this structure before returning (caller can safely destroy the object).

Returns

A pointer to a newly instantiated plugin or NULL if an error occurred during creation.

5.3.5.2 NDDS_Transport_TCPv4_create()

```
NDDS_Transport_Plugin * NDDS_Transport_TCPv4_create (
             NDDS_Transport_Address_t * default_network_address_out,
            const struct  DDS_PropertyQosPolicy * property_in )
```

Creates an instance of a TCP Transport Plugin, using PropertyQosPolicy.

For all the properties that are not specified, the default value is assumed (as described in **Configure TCP Transport with Property QoS Policy** (p. 22)).

Parameters

default_network_address_out	<<***out***>> Unused parameter
property_in	<<***in***>> The DDS_PropertyQoSPolicy object containing the transport properties.

Returns

A pointer to a newly instantiated plugin or NULL if an error occurred during creation.

5.3.5.3 NDDS_Transport_TCPv4_Plugin_stringToTransportAddress()

```
RTIBool NDDS_Transport_TCPv4_Plugin_stringToTransportAddress (
             NDDS_Transport_Address_t * address_out,
            const char * address_in,
            RTIBool acceptHostName_in )
```

Converts a string into a transport address suitable to be used with the TCP transport.

This method will convert a string in the form **host:port** into a **NDDS_Transport_Address_t** structure.

If the 'port' component is not specified, it will assign the default value as described by the constant **NDDS_↩ TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER** (p. 12).

Parameters

address_out	<<***out***>> Transport address to initialize
address_in	<<***in***>> the NULL-terminated string containing the address to parse
acceptHostName_in	<<***in***>> Set it to RTI_TRUE to allow this function to resolve the host component of the string address through a DNS lookup; set it to RTI_FALSE to allow only IPv4 addresses in the 'host' part of the address string.

Returns

RTI_TRUE if success or RTI_FALSE if an error occurred during the conversion.

5.4 Version

Version utilities.

Typedefs

- typedef struct **DDS_ProductVersion_t NDDS_Transport_TCPv4_LibraryVersion**
  The TCP transport library version.

Functions

- const **NDDS_Transport_TCPv4_LibraryVersion** * **NDDS_Transport_TCPv4_get_library_version** (void)
  Get the TCP transport library version.
- const char * **NDDS_Transport_TCPv4_get_build_version_string** (void)
  Get the TCP transport library build version.

5.4.1 Detailed Description

Version utilities.

5.4.2 Typedef Documentation

5.4.2.1 NDDS_Transport_TCPv4_LibraryVersion

```
typedef struct DDS_ProductVersion_t NDDS_Transport_TCPv4_LibraryVersion
```

The TCP transport library version.

5.4.3 Function Documentation

5.4.3.1 NDDS_Transport_TCPv4_get_library_version()

```
const NDDS_Transport_TCPv4_LibraryVersion * NDDS_Transport_TCPv4_get_library_version (
            void )
```

Get the TCP transport library version.

5.4.3.2 NDDS_Transport_TCPv4_get_build_version_string()

```
const char * NDDS_Transport_TCPv4_get_build_version_string (
            void )
```

Get the TCP transport library build version.

5.5 TCP Transport Initial Peers

This section describes how to specify the TCP transport initial peers.

This section describes how to specify the TCP transport initial peers.

With the TCP transport, the addresses of the initial peers (NDDS_DISCOVERY_PEERS) that will be contacted during the discovery process have the following format:

- For WAN communication: tcpv4_wan://<IP address or hostname>:<port>
- For LAN communication: tcpv4_lan://<IP address or hostname>:<port>

For example: setenv NDDS_DISCOVERY_PEERS tcpv4_wan://10.10.1.165:7400,tcpv4_wan://10.10.1.111↩
:7400,tcpv4_lan://192.168.1.1:7500

When the TCP transport is configured for LAN communication (with the **NDDS_Transport_Property_t::classid** property), the IP address is the LAN address of the peer and the port is the server port used by the transport (the **NDDS_↩
Transport_TCPv4_Property_t::server_bind_port** (p. 81) property).

When the TCP transport is configured for WAN communication (with the **NDDS_Transport_Property_t::classid** property), the IP address is the WAN or public address of the peer and the port is the public port that is used to forward traffic to the server port in the TCP transport.

5.6 Configure TCP Transport with Property QoS Policy

Predefined properties that can be used to configure the TCP Transport plugin through the Property QoS Policy of the DomainParticipant.

Predefined properties that can be used to configure the TCP Transport plugin through the Property QoS Policy of the DomainParticipant.

Table 5.8 ***Properties for Registering the TCP Transport***

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". In the following examples, <*TCP_prefix*> indicates the string that is used as a prefix in the property names for all the settings that are related to this TCP transport plugin. <*TCP_prefix*> must begin with "dds.transport." (such as "dds.transport.TCPv4.tcp1").	**YES**
<*TCP_prefix*>.library	Must set to **"nddstransporttcp"**. The library **"libnddstransporttcp.so"** (on Linux systems), **"nddstransporttcp.dylib"** (on macOS systems), or **"nddstransporttcp.dll"** (on Windows systems) 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**
<*TCP_prefix*>.create_function	Must be set to **"NDDS_Transport_TCPv4_create"** .	**YES**
<*TCP_prefix*>.aliases	Used to register the transport plugin returned by **NDDS_Transport_TCPv4_create** (p. 20) (as specified by <***TCP_prefix***>**.create_function**) to the DomainParticipant. Refer to **aliases_in** parameter in **NDDS_Transport_Support_register_transport** for details. Aliases should be specified as comma-separated strings, with each comma delimiting an alias. If it is not specified, <*TCP_prefix*> is used as the default alias for the plugin.	NO

Property Name	Description	Required?
<*TCP_prefix*>.parent.classid	Must be set to one of the following values: • NDDS_TRANSPORT_↩ CLASSID_TCPV4_LAN for TCP communication within a LAN. • NDDS_TRANSPORT_↩ CLASSID_TCPV4_WAN for TCP communication across LANs and firewalls. Default: NDDS_TRANSPORT_↩ CLASSID_TCPV4_LAN	NO
<*TCP_prefix*>.parent.gather_↩ send_buffer_count_max	See **NDDS_Transport_Property↩ _t::gather_send_buffer_count_↩ max** Default: 128	NO
<*TCP_prefix*>.parent.message_↩ size_max	See **NDDS_Transport_Property↩ _t::message_size_max** Default: 9216	NO
<*TCP_prefix*>.parent.allow_↩ interfaces	See **NDDS_Transport_Property↩ _t::allow_interfaces_list** and **NDDS_Transport_Property_t↩ ::allow_interfaces_list_length**. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example, "127.0.0.1,eth0" Default: All available interfaces are used.	NO
<*TCP_prefix*>.parent.deny_↩ interfaces	See **NDDS_Transport_Property↩ _t::deny_interfaces_list** and **NDDS_Transport_Property_t↩ ::deny_interfaces_list_length**. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example, "127.0.0.1,eth0" Default: No interfaces are denied.	NO
<*TCP_prefix*>.parent.thread_↩ name_prefix	See **NDDS_Transport_Property↩ _t::thread_name_prefix**	NO

Property Name	Description	Required?
<*TCP_prefix*>.send_socket_↩ buffer_size	See **NDDS_Transport_TCPv4_↩ Property_t::send_socket_buffer↩ _size** (p. 78) Default: -1 (setsockopt() (or equivalent) will not be called to size the send buffer of the socket)	NO
<*TCP_prefix*>.recv_socket_↩ buffer_size	See **NDDS_Transport_TCPv4_↩ Property_t::recv_socket_buffer↩ _size** (p. 78) Default: -1 (setsockopt() (or equivalent) will not be called to size the send buffer of the socket)	NO
<*TCP_prefix*>.ignore_loopback_↩ interface	See **NDDS_Transport_TCPv4_↩ Property_t::ignore_loopback_↩ interface** (p. 78) Default: 1	NO
<*TCP_prefix*>.ignore_↩ nonrunning_interfaces	See **NDDS_Transport_TCPv4_↩ Property_t::ignore_nonrunning↩ _interfaces** (p. 78) Default: 1	NO
<*TCP_prefix*>.transport_priority↩ _mask	See **NDDS_Transport_TCPv4_↩ Property_t::transport_priority_↩ mask** (p. 79) Default: 0	NO
<*TCP_prefix*>.transport_priority↩ _mapping_low	See **NDDS_Transport_TCPv4_↩ Property_t::transport_priority_↩ mapping_low** (p. 79) Default: 0	NO
<*TCP_prefix*>.transport_priority↩ _mapping_high	See **NDDS_Transport_TCPv4_↩ Property_t::transport_priority_↩ mapping_high** (p. 80) Default: 0xFF	NO
<*TCP_prefix*>.server_socket_↩ backlog	See **NDDS_Transport_TCPv4↩ _Property_t::server_socket_↩ backlog** (p. 81) Default: 5	NO

Property Name	Description	Required?
<*TCP_prefix*>.public_address	See **NDDS_Transport_TCPv4_↩ Property_t::public_address** (p. 81) Public IP address and port (WAN address and port) associated with the transport instantiation. The address and port are separated with a colon. For example: 10.10.9.10:4567 This field is used only when parent.classid is set to NDDS_↩ TRANSPORT_CLASSID_TCPV4↩ _WAN. The public address and port are necessary to support communication over WAN that involves Network Address Translators (NATs). Typically, the address is the public address of the IP router that provides access to the WAN. The port is the IP router port that is used to reach the private server_bind_port inside the LAN from the outside. This value is expressed as a string in the form: ip[:port], where ip represents the IPv4 address and port is the external port number of the router. Note that host names are not allowed in the public_address because they may resolve to an internet address that is not what you want (i.e., 'localhost' may map to your local IP or to 127.0.0.1). Note: If you are using an asymmetric configuration, public_address does not have to be set for the non-public peer.	**YES** (For WAN classid); see Note at left.
<*TCP_prefix*>.server_bind_port	See **NDDS_Transport_TCPv4↩ _Property_t::server_bind_port** (p. 81) Default: 7400	NO
<*TCP_prefix*>.read_buffer_↩ allocation.initial_count	See **NDDS_Transport_TCPv4↩ _Property_t::read_buffer_↩ allocation** (p. 82) Default: 2	NO

Property Name	Description	Required?
<*TCP_prefix*>.read_buffer_↩ allocation.max_count	See **NDDS_Transport_TCPv4**↩ **_Property_t::read_buffer_**↩ **allocation** (p. 82) Default: -1 (unlimited)	NO
<*TCP_prefix*>.read_buffer_↩ allocation.incremental_count	See **NDDS_Transport_TCPv4**↩ **_Property_t::read_buffer_**↩ **allocation** (p. 82) Default: -1 (number of buffers will keep doubling on each allocation until it reaches max_count)	NO
<*TCP_prefix*>.write_buffer_↩ allocation.initial_count	See **NDDS_Transport_TCPv4**↩ **_Property_t::write_buffer_**↩ **allocation** (p. 82) Default: 4	NO
<*TCP_prefix*>.write_buffer_↩ allocation.max_count	See **NDDS_Transport_TCPv4**↩ **_Property_t::write_buffer_**↩ **allocation** (p. 82) Default: 1000	NO
<*TCP_prefix*>.write_buffer_↩ allocation.incremental_count	See **NDDS_Transport_TCPv4**↩ **_Property_t::write_buffer_**↩ **allocation** (p. 82) Default: 10	NO
<*TCP_prefix*>.shared_write_↩ buffer_allocation	See **NDDS_Transport_TCPv4**↩ **_Property_t::shared_write_**↩ **buffer_allocation** (p. 82) Default: 0	NO
<*TCP_prefix*>.control_buffer_↩ allocation.initial_count	See **NDDS_Transport_TCPv4**↩ **_Property_t::control_buffer_**↩ **allocation** (p. 83) Default: 2	NO
<*TCP_prefix*>.control_buffer_↩ allocation.max_count	See **NDDS_Transport_TCPv4**↩ **_Property_t::control_buffer_**↩ **allocation** (p. 83) Default: -1 (unlimited)	NO
<*TCP_prefix*>.control_buffer_↩ allocation.incremental_count	See **NDDS_Transport_TCPv4**↩ **_Property_t::control_buffer_**↩ **allocation** (p. 83) Default: -1 (number of buffers will keep doubling on each allocation until it reaches max_count)	NO

Property Name	Description	Required?
<*TCP_prefix*>.control_message_allocation.initial_count	See **NDDS_Transport_TCPv4_Property_t::control_message_allocation** (p. 83) Default: 2	NO
<*TCP_prefix*>.control_message_allocation.max_count	See **NDDS_Transport_TCPv4_Property_t::control_message_allocation** (p. 83) Default: -1 (unlimited)	NO
<*TCP_prefix*>.control_message_allocation.incremental_count	See **NDDS_Transport_TCPv4_Property_t::control_message_allocation** (p. 83) Default: -1 (number of buffers will keep doubling on each allocation until it reaches max_count)	NO
<*TCP_prefix*>.control_attribute_allocation.initial_count	See **NDDS_Transport_TCPv4_Property_t::control_attribute_allocation** (p. 83) Default: 2	NO
<*TCP_prefix*>.control_attribute_allocation.max_count	See **NDDS_Transport_TCPv4_Property_t::control_attribute_allocation** (p. 83) Default: -1 (unlimited)	NO
<*TCP_prefix*>.control_attribute_allocation.incremental_count	See **NDDS_Transport_TCPv4_Property_t::control_attribute_allocation** (p. 83) Default: -1 (number of buffers will keep doubling on each allocation until it reaches max_count)	NO
<*TCP_prefix*>.force_asynchronous_send	See **NDDS_Transport_TCPv4_Property_t::force_asynchronous_send** (p. 83) Default: 0	NO
<*TCP_prefix*>.max_packet_size	See **NDDS_Transport_TCPv4_Property_t::max_packet_size** (p. 84) Default: -1 (OS default value)	NO
<*TCP_prefix*>.enable_keep_alive	See **NDDS_Transport_TCPv4_Property_t::enable_keep_alive** (p. 85) Default: 0	NO

Property Name	Description	Required?
<*TCP_prefix*>.keep_alive_time	See **NDDS_Transport_TCPv4_Property_t::keep_alive_time** (p. 85) Default: -1 (OS default value)	NO
<*TCP_prefix*>.keep_alive_interval	See **NDDS_Transport_TCPv4_Property_t::keep_alive_interval** (p. 85) Default: -1 (OS default value)	NO
<*TCP_prefix*>.keep_alive_retry_count	See **NDDS_Transport_TCPv4_Property_t::keep_alive_retry_count** (p. 85) Default: -1 (OS default value)	NO
<*TCP_prefix*>.disable_nagle	See **NDDS_Transport_TCPv4_Property_t::disable_nagle** (p. 86) Default: 0	NO
<*TCP_prefix*>.logging_verbosity_bitmap	See **NDDS_Transport_TCPv4_Property_t::logging_verbosity_bitmap** (p. 86) Default: -1 (do not change default verbosity)	NO
<*TCP_prefix*>.outstanding_connection_cookies	See **NDDS_Transport_TCPv4_Property_t::outstanding_connection_cookies** (p. 88) Default: 100	NO
<*TCP_prefix*>.outstanding_connection_cookies_life_span	See **NDDS_Transport_TCPv4_Property_t::outstanding_connection_cookies_life_span** (p. 88) Default: -1, which means an unlimited amount of time (effectively disabling the feature)	NO
<*TCP_prefix*>.send_max_wait_sec	See **NDDS_Transport_TCPv4_Property_t::send_max_wait_sec** (p. 89) Default: 3 seconds	NO

Property Name	Description	Required?
<*TCP_prefix*>.socket_↩ monitoring_kind	See **NDDS_Transport_TCPv4_↩ Property_t::socket_monitoring↩ _kind** (p. 87) Possible values: • SELECT • WINDOWS_IOCP • WINDOWS_↩ WAITFORMULTIPLEOBJECT Default: SELECT	NO
<*TCP_prefix*>.windows_iocp.↩ thread_pool_size	See **NDDS_Transport_TCPv4↩ _Property_t::windows_iocp_↩ settings** (p. 88) Default: 2	NO
<*TCP_prefix*>.windows_iocp.↩ concurrency_value	See **NDDS_Transport_TCPv4↩ _Property_t::windows_iocp_↩ settings** (p. 88) Default: 1	NO
<*TCP_prefix*>.disable_interface_↩ tracking	See **NDDS_Transport_TCPv4_↩ Property_t::disable_interface_↩ tracking** (p. 90) Default: 0	NO
<*TCP_prefix*>.force_interface_↩ poll_detection	See **NDDS_Transport_TCPv4↩ _Property_t::force_interface_↩ poll_detection** (p. 80) Default: 0	NO
<*TCP_prefix*>.interface_poll_↩ period	See **NDDS_Transport_TCPv4_↩ Property_t::interface_poll_period** (p. 80) Default: 500 milliseconds	NO
<*TCP_prefix*>.tls.verify.ca_file	See **RTITLS_Verification::ca_file** (p. 98) Default: NULL	NO
<*TCP_prefix*>.tls.verify.ca_path	See **RTITLS_Verification::ca_path** (p. 98) Default: NULL	NO
<*TCP_prefix*>.tls.verify.verify_↩ depth	See **RTITLS_Verification::verify↩ _depth** (p. 99) Default: -1 (no limit)	NO

Property Name	Description	Required?
<*TCP_prefix*>.tls.verify.crl_file	See **RTITLS_Verification::crl_file** (p. 99) Default: NULL	NO
<*TCP_prefix*>.tls.cipher.cipher_list	See **RTITLS_Ciphers::cipher_list** (p. 92) Default: NULL	NO
<*TCP_prefix*>.tls.cipher.↩ ciphersuites	See **RTITLS_Ciphers↩ ::ciphersuites** (p. 92)	NO
<*TCP_prefix*>.tls.cipher.dh_↩ param_files	See **RTITLS_Ciphers::dh_↩ param_files** (p. 93) Default: NULL	NO
<*TCP_prefix*>.tls.cipher.engine_id	See **RTITLS_Ciphers::engine_id** (p. 93) Default: NULL	NO
<*TCP_prefix*>.tls.identity.↩ certificate_chain	See **RTITLS_Identity::certificate↩ _chain** (p. 95) Default: NULL	NO
<*TCP_prefix*>.tls.identity.↩ certificate_chain_file	See **RTITLS_Identity::certificate↩ _chain_file** (p. 95) Default: NULL	NO
<*TCP_prefix*>.tls.identity.private↩ _key_password	See **RTITLS_Identity::private_↩ key_password** (p. 95) Default: NULL (no password)	NO
<*TCP_prefix*>.tls.identity.private↩ _key	See **RTITLS_Identity::private_key** (p. 96) Default: NULL	NO
<*TCP_prefix*>.tls.identity.private↩ _key_file	See **RTITLS_Identity::private_↩ key_file** (p. 96) Default: NULL	NO
<*TCP_prefix*>.tls.identity.rsa_↩ private_key	See **RTITLS_Identity::rsa_↩ private_key** (p. 96) Default: NULL	NO
<*TCP_prefix*>.tls.identity.rsa_↩ private_key_file	See **RTITLS_Identity::rsa_↩ private_key_file** (p. 96) Default: NULL	NO

5.7 TCP Transport Examples

TCP Transport Examples.

Modules

- **C Example of TCP Transport with PropertiesQosPolicy Configuration**

  TCP Transport example with PropertiesQosPolicy configuration.

5.7.1 Detailed Description

TCP Transport Examples.

5.8 RTI TLS Support

OpenSSL configuration interfaces and definitions.
TLS Support is included with some RTI purchases and must be downloaded and installed separately.

Data Structures

- struct **RTITLS_Verification**

  Set of TLS properties for certificate authorities (CAs) and verification.
- struct **RTITLS_Identity**

  Set of TLS properties for identity.
- struct **RTITLS_DHParamFile**

  Name of a Diffie-Helman (DH) key file and the length of the contained key in bits.
- struct **RTITLS_Ciphers**

  Set of TLS properties for cipher configuration.
- struct **RTITLS_OpenSSL_Configuration**

  Full set of TLS properties.

Macros

- #define **RTITLS_VERIFY_DEFAULT**

  Use this to initialize a RTITLS_Verfication structure.
- #define **RTITLS_IDENTITY_DEFAULT**

  *Use this to initialize a **RTITLS_Identity** (p. 94) structure.*
- #define **RTITLS_CIPHER_LIST_DEFAULT** "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH"

  Cipher list string for default channel (encrypted)
- #define **RTITLS_CIPHER_LIST_ENCRYPT_HIGH** "AES:ALL:!aNULL:!eNULL:!LOW:!EXP:+RC4:@STRENGTH"

  Cipher list string for default channel (encrypted, no low-strength)
- #define **RTITLS_CIPHER_LIST_UNENCRYPTED** "aNULL"

  Cipher list string for authentication-only channel (no encryption)
- #define **RTITLS_CIPHER_DEFAULT**

  *Use this to initialize a **RTITLS_Ciphers** (p. 92) structure.*
- #define **RTITLS_OPENSSL_CONFIGURATION_DEFAULT**

  *Use this to initialize a **RTITLS_OpenSSL_Configuration** (p. 97) structure.*

Typedefs

- typedef int(∗ **RTITLS_Verify_Callback**) (int preverify_ok, X509_STORE_CTX ∗x509_ctx)

  Callback used to verify peer certificates.

Functions

- void **RTITLS_thread_exit** (void)

  clean up OpenSSL resources for current thread (call before exit)
- int **RTITLS_default_verify_callback** (int ok, X509_STORE_CTX ∗store)

  Default verify callback: log errors when verification fails.
- int **RTITLS_verbose_verify_callback** (int ok, X509_STORE_CTX ∗store)

  Verbose verify callback: log information about successful verification as well as errors when verification fails.

5.8.1 Detailed Description

OpenSSL configuration interfaces and definitions.
TLS Support is included with some RTI purchases and must be downloaded and installed separately.

5.8.2 Macro Definition Documentation

5.8.2.1 RTITLS_VERIFY_DEFAULT

```
#define RTITLS_VERIFY_DEFAULT
```

Value:

```
    { \
    NULL, NULL, NULL, /* ca_file, ca_path, ca */ \
    -1, /* verify_depth (no depth limit) */ \
    NULL, /* callback (use default verify callback) */ \
    NULL /* crl_file */ }
```

Use this to initialize a RTITLS_Verfication structure.

5.8.2.2 RTITLS_IDENTITY_DEFAULT

```
#define RTITLS_IDENTITY_DEFAULT
```

Value:

```
    { \
    NULL, /* certificate_chain */ \
    NULL, /* certificate_chain_file */ \
    NULL, /* private_key_password */ \
    NULL, /* private_key */ \
    NULL, /* private_key_file */ \
    NULL, /* rsa_private_key */ \
    NULL /* rsa_private_key_file */ }
```

Use this to initialize a **RTITLS_Identity** (p. 94) structure.

5.8.2.3 RTITLS_CIPHER_LIST_DEFAULT

```
#define RTITLS_CIPHER_LIST_DEFAULT "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH"
```

Cipher list string for default channel (encrypted)

5.8.2.4 RTITLS_CIPHER_LIST_ENCRYPT_HIGH

```
#define RTITLS_CIPHER_LIST_ENCRYPT_HIGH "AES:ALL:!aNULL:!eNULL:!LOW:!EXP:+RC4:@STRENGTH"
```

Cipher list string for default channel (encrypted, no low-strength)

5.8.2.5 RTITLS_CIPHER_LIST_UNENCRYPTED

```
#define RTITLS_CIPHER_LIST_UNENCRYPTED "aNULL"
```

Cipher list string for authentication-only channel (no encryption)

5.8.2.6 RTITLS_CIPHER_DEFAULT

```
#define RTITLS_CIPHER_DEFAULT
```

Value:

```
{ \
NULL, /* cipher_list */ \
NULL, /* ciphersuites */ \
0, NULL, /* dh_param_files_length, dh_param_files (no DH params) */ \
NULL, /* engine_id (no engine) */ \
0, NULL, NULL, /* engine_pre_cmd_length, engine_pre_cmd_names, engine_pre_cmd_parameters */ \
0, NULL, NULL /* engine_post_cmd_length, engine_post_cmd_names, engeine_post_cmd_parameters */ }
```

Use this to initialize a **RTITLS_Ciphers** (p. 92) structure.

5.8.2.7 RTITLS_OPENSSL_CONFIGURATION_DEFAULT

```
#define RTITLS_OPENSSL_CONFIGURATION_DEFAULT
```

Value:

```
{ \
RTITLS_VERIFY_DEFAULT, /* verify */ \
RTITLS_IDENTITY_DEFAULT, /* identity */ \
RTITLS_CIPHER_DEFAULT, /* cipher */ \
RTITLS_RENEGOTIATE_DEFAULT /* renegotiate */ }
```

Use this to initialize a **RTITLS_OpenSSL_Configuration** (p. 97) structure.

5.8.3 Typedef Documentation

5.8.3.1 RTITLS_Verify_Callback

```
typedef int(* RTITLS_Verify_Callback) (int preverify_ok, X509_STORE_CTX *x509_ctx)
```

Callback used to verify peer certificates.

See the OpenSSL manual page for SSL_CTX_set_verify for more information.

5.8.4 Function Documentation

5.8.4.1 RTITLS_thread_exit()

```
void RTITLS_thread_exit (
            void  )
```

clean up OpenSSL resources for current thread (call before exit)

5.8.4.2 RTITLS_default_verify_callback()

```
int RTITLS_default_verify_callback (
            int ok,
            X509_STORE_CTX * store )
```

Default verify callback: log errors when verification fails.

See the OpenSSL manual page for SSL_CTX_set_verify for more information.

5.8.4.3 RTITLS_verbose_verify_callback()

```
int RTITLS_verbose_verify_callback (
            int ok,
            X509_STORE_CTX * store )
```

Verbose verify callback: log information about successful verification as well as errors when verification fails.

See the OpenSSL manual page for SSL_CTX_set_verify for more information.

5.9 C Example of TCP Transport with PropertiesQosPolicy Configuration

TCP Transport example with PropertiesQosPolicy configuration.

Modules

- **HelloWorld.idl**
- **HelloCommon.h**
- **HelloWorld.c**
- **HelloWorld_publisher.c**
- **HelloWorld_subscriber.c**
- **HelloWorldPlugin.c**
- **HelloWorldSupport.c**

5.9.1 Detailed Description

TCP Transport example with PropertiesQosPolicy configuration.

See README.txt for details.

[$(NDDSHOME)/example/C/helloWorldTCP/README.txt]

```
=======================================
C Hello World Example with TCP Transport
=======================================
This example is a modified version of the helloWorld example that can
be generated using rtiddsgen with the -example command-line option.
The original example was modified to use the TCP transport instead of
the built-in transports (UDPv4 and shared memory).
This example uses the DDS properties to configure the application to
use the TCP Transport.
Purpose
=======
This example shows how to create an instance of the TCP transport
plugin and tell Connext DDS to use it as the only communication mechanism
between peers.
How to Compile
==============
Make sure the environment variable NDDSHOME is defined before compiling
the example.
Linux and macOS systems
-----------------
Before compiling, make sure that the desired version of compiler and linker
are in your PATH environment variable.
Then compile by typing:
make -f make/makefile_HelloWorld_<architecture>
Windows systems
---------------
Open the solution file for your architecture with Visual Studio. Build the
two projects, HelloWorld_publisher and HelloWorld_subscriber, with the
Win32 Release setting (or Win32 Debug settings if you prefer).
How to Run
==========
1.  In your library search path, you MUST have the location of
    the transport plugin shared library, because the applications
    will load the shared object dynamically when they start.
    For Linux and macOS systems, you need to add the directory:
            $NDDSHOME/lib/<architecture>
    to your LD_LIBRARY_PATH.
    For Windows systems, you need to add the the directory:
            %NDDSHOME%\lib\<architecture>
    to your PATH environment variable.
2. To test a WAN scenario, you need to know the public IP address
   of each peer and have the firewall correctly configured.
   Refer to the Connext DDS User's Manual for more details.
```

```
   For a LAN scenario or to run the publisher and subscriber on the
   same machine, see next section.

3. On one machine, run the publisher:

   On Linux and macOS systems:

   export NDDS_DISCOVERY_PEERS=tcpv4_wan://<subscriber_public_ip:port>
   ./objs/<arch>/HelloWorld_publisher -a <public_ip:port> -p <bind_port>

   On Windows systems:

   set NDDS_DISCOVERY_PEERS=tcpv4_wan://<subscriber_public_ip:port>
   objs\<arch>\HelloWorld_publisher.exe -a <public_ip:port> -p <bind_port>

   Replace <subscriber_public_ip:port> with the public IP and port of
   the remote peer that will run the subscriber.
   Replace <public_ip:port> with your public IP and port that are
   exposed to the outside network.
   Replace <bind_port> with the port number on your local machine
   to listen for incoming connections.
   See the example below for a real scenario.

   You can add the command-line option '-d <domain>' to specify a
   domain ID (if unset, domain ID 0 is used).

4. On another machine, run the subscriber:

   On Linux and macOS systems:

   export NDDS_DISCOVERY_PEERS=tcpv4_wan://<publisher_public_ip:port>
   ./objs/<arch>/HelloWorld_subscriber -a <public_ip:port> -p <bind_port>

   On Windows systems:

   set NDDS_DISCOVERY_PEERS=tcpv4_wan://<publisher_public_ip:port>
   objs\<arch>\HelloWorld_subscriber.exe -a <public_ip:port> -p <bind_port>

   Similarly, <publisher_public_ip:port> is the public IP and port of
   the publisher.


As an example, consider the following configuration:

PUBLISHER:
                      +-------------+
    +-----+           | Router &    |
    | pub |-----------| Firewall    |
    +-----+           |             | public ip: 99.88.77.66
   192.168.1.10:7400  +-------------+ fwd port: 5555

   - The publisher application is running on the machine
     at 192.168.1.10
   - The publisher application is using as bind port: 7400
   - The router has port forwarding configured as:
       src port: 5555, destination: 192.168.1.10:7400
   - The public IP of this network is 99.88.77.66

SUBSCRIBER:
                      +-------------+
    +-----+           | Router &    |
    | sub |-----------| Firewall    |
    +-----+           |             | public ip: 11.22.33.44
   192.168.5.20:7400  +-------------+ fwd port: 4444

   - The subscriber application is running on the machine
     at 192.168.5.20
   - The subscriber application is using as bind port: 7400
   - The router has port forwarding configured as:
       src port: 4444, destination: 192.168.5.20:7400
   - The public IP of this network is 11.22.33.44

In this case, use the following commands to start the example:

PUBLISHER:
  On Linux and macOS systems:

    export NDDS_DISCOVERY_PEERS=tcpv4_wan://11.22.33.44:4444
    ./objs/<arch>/HelloWorld_publisher -a 99.88.77.66:5555 -p 7400
```

```
  On Windows systems:

    set NDDS_DISCOVERY_PEERS=tcpv4_wan://11.22.33.44:4444
    objs\<arch>\HelloWorld_publisher -a 99.88.77.66:5555 -p 7400

SUBSCRIBER:
  On Linux and macOS systems:

    export NDDS_DISCOVERY_PEERS=tcpv4_wan://99.88.77.66:5555
    ./objs/<arch>/HelloWorld_subscriber -a 11.22.33.44:4444 -p 7400

  On Windows systems:

    set NDDS_DISCOVERY_PEERS=tcpv4_wan://99.88.77.66:5555
    objs\<arch>\HelloWorld_subscriber -a 11.22.33.44:4444 -p 7400

Notes:
    You do not need to set the environment variable
    NDDS_DISCOVERY_PEERS on both the publisher and subscriber; the
    discovery protocol will take care of propagating the public location
    to the other node as soon as communication is established, but keep
    in mind that it may take a longer time for discovery to complete
    this way.

    You CAN use a symbolic name instead of a numeric IP address
    when you specify the NDDS_DISCOVERY_PEERS value, but you MUST
    use a numeric IP address when you specify the public IP address of
    the network (the '-a' command-line option).

Running within a LAN
====================

When running within a LAN, you do not need to
specify the '-a' parameter, because the transport will automatically
use all the available network interface for communication.

Also remember that the WAN and LAN have two different transport IDs,
so when you specify the locators, you need to use 'tcpv4_lan'
instead of 'tcpv4_wan.'

Even if the machines are within the same LAN, you can still use the
WAN transport ID, but you will need to specify the '-a' parameter.


Running on the same machine
===========================

When running on the same host, make sure you use different <bind_port>
for your publisher and subscriber applications.

For example, to run the publisher and subscriber on the same machine
using the LAN transport ID:

Publisher:
 On Linux and macOS systems:
   export NDDS_DISCOVERY_PEERS=tcpv4_lan://localhost:7401
   ./objs/<arch>/HelloWorld_publisher -p 7400

 On Windows systems:
   set NDDS_DISCOVERY_PEERS=tcpv4_lan://localhost:7401
   objs\<arch>\HelloWorld_publisher -p 7400

Subscriber:
 On Linux and macOS systems:
   export NDDS_DISCOVERY_PEERS=tcpv4_lan://localhost:7400
   ./objs/<arch>/HelloWorld_subscriber -p 7401

 On Windows systems:
   set NDDS_DISCOVERY_PEERS=tcpv4_lan://localhost:7400
   objs\<arch>\HelloWorld_subscriber -p 7401


Running with TLS enabled
========================

In order to use TCP transport with secure communication in Connext DDS,
you will need to install the optional packages, RTI TLS
support and the OpenSSL library.

In its simplest form, only two properties are required to enable TLS secure
communication between different peers:
```

```
    tls.verify.ca_file - the certificate file
    tls.identity.certificate_chain_file - the certificate chain file

The two parameters are both required to enable TLS.
In this example, you can set the two above parameters with the following two
command-line arguments:
    --tls-cert <ca_cert>
    --tls-chain <cert_chain>

When the two parameters are specified to valid files, the transport will
run using TLS enabled.

Example certificates for two peers are included in dds_security/cert/tls_rsa01.

The internal behavior of the transport plugin will be exactly the same
as the case with security disabled, except that all the sockets used rely on
OpenSSL for encryption.

IMPORTANT NOTES:
1. If you use TLS, you must set the transport's class ID to one of the following
   constants in your user application:
        NDDS_TRANSPORT_CLASSID_TLSV4_WAN
        NDDS_TRANSPORT_CLASSID_TLSV4_LAN
   This example application automatically sets this class ID for you
   based on the command line arguments.
2. Remember also that you MUST use the prefix 'tlsv4_lan://', or 'tlsv4_wan://'
   instead of 'tcpv4_lan://' and 'tcpv4_wan://' when you define your initial
   peer host.
   ALSO NOTE THAT TCP AND TLS ARE NOT INTEROPERABLE!
   One peer implementing secure connection can never talk with another peer
   that uses the un-encrypted TCP connection.
3. In order to break the dependency between the TCP transport library and the
   OpenSSL library, when TLS security is enabled:
        - If you are linking dynamically, the transport will load the following
        library:
            On Linux and macOS Systems:     libnddstls.so (release version)
                                            libnddstlsd.so (debug version)
            On Windows:                     nddstls.dll (release version)
                                            nddstlsd.dll (debug version)
            The NDDSTLS library will then dynamically load OpenSSL during
            its initialization.
            YOU MUST ENSURE THAT THE ABOVE LIBRARIES ARE PRESENT IN YOUR SEARCH
            PATH BEFORE STARTING YOUR APPLICATION (if your application enables
            TLS). This requirement is not needed if your application does not
            use TLS.
        - If you are linking statically, you will need to make the following
        manual modifications to the example:
            1. Uncomment the USE_TLS define in src/HelloCommon.h.
            2. Manually add the following libraries to the makefile/vcxproj file
                On Linux and macOS systems:
                    $(NDDSHOME)/lib/<architecture>/libnddstlsz.a (or libnddstlszd.a for debug)
                    $(NDDSHOME)/third-party/openssl-<version>/<architecture>/<release or debug>/lib/libssl.a
                    $(NDDSHOME)/third-party/openssl-<version>/<architecture>/<release or debug>/lib/libcrypto.a
                On Windows systems:
                    $(NDDSHOME)\lib\<architecture>\nddstlsz.lib (or nddstlszd.lib for debug)
                    $(NDDSHOME)\third-party\openssl-<version>\<architecture>\<static_release or
       static_debug>\lib\libssl.lib
                    $(NDDSHOME)\third-party\openssl-<version>\<architecture>\<static_release or
       static_debug>\lib\libcrypto.lib
                    crypt32.lib (Windows built-in library)
4. In symmetric communication scenarios, any Connext DDS
   application can initiate TCP connections with other applications.
   In asymmetric communication scenarios, TCP connections can be
   initiated only by one Connext DDS instance. For
   security reasons, any incoming connection from the other instance is
   not allowed.
   By running the example as stated above with TLS enabled, it is runing in
   symmetric mode.
   To run in asymmetric mode, leave the NDDS_DISCOVERY_PEERS unset on
   the server side and don't provide the "-a" and "-p" command line
   arguments on the client side (because the client side has no publicly
   reachable address:port).
```

5.10 HelloWorld.idl

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorld.idl]

```
struct HelloWorld {
        string<128> msg;
};
```

5.11 HelloCommon.h

[$(NDDSHOME)/example/C/helloWorldTCP/HelloCommon.h]

```
/* Common code between pub & sub */
#include "ndds/osapi/osapi_utility.h"
#if (defined(_WIN32) || defined(WIN32))
#define RTI_STRCASECMP      _stricmp
#define RTI_STRNCASECMP     _strnicmp
#define RTI_STRTOULL        _strtoui64
#define RTI_STRTOUL         strtoul
#define RTI_STRNCMP         strncmp
#else
#define RTI_STRCASECMP      strcasecmp
#define RTI_STRNCASECMP     strncasecmp
#define RTI_STRTOULL        strtoull
#define RTI_STRTOUL         strtoul
#define RTI_STRNCMP         strncmp
#endif
/* define this to enable TLS */
/* #define USE_TLS */
#ifdef RTI_STATIC
  #include "ndds/transport_tcp/transport_tcp_tcpv4.h"
  #ifdef USE_TLS
    #include "ndds/rtitls/rtitls_connection.h"
  #endif
#else
  #define TRANSPORT_LIB_NAME "nddstransporttcp"
  #define TRANSPORT_LIB_NAME_DBG "nddstransporttcpd"
#endif
#define DEFAULT_SERVER_PORT         7400
struct CmdLineArgs {
    int domain_id;
    int sample_count;
    const char * addr;
    int port;
    int verbose;
    RTIBool asyncSend;
    RTIBool debug;
    RTIBool tls;
    const char *tlsCert;
    const char *tlsChain;
};
static void usage(const char *appName) {
    printf("Usage: appName [arguments]\n");
    printf("Where arguments are:\n");
    printf("  -h | --help                "
           "Shows this page\n");
    printf("  -v | --verbose             "
           "Increase output verbosity\n");
    printf("  -s | --sampleCount <num>   "
           "Sets number of loops to perform [default=unlimited]\n");
    printf("  -d | --domain <domainID>   "
           "Sets the DDS domain ID [default=0]\n");
    printf("  --debug                    "
           "Use the debug version of the plugin\n");
    printf("  --async                    "
           "Use asynchronous send [default=use synchronous send]\n");
    printf("  -a | --address <ip:port>   "
           "Sets the public IP and port for the TCP transport\n");
    printf("  -p | --port <port>         "
           "Enable server socket bound at specific port (default=disabled)\n");
    printf("  --tls-cert <ca_cert>       "
           "Sets certificate of authority used in TLS configuration\n");
    printf("  --tls-chain <cert_chain>   "
           "Sets certificate chain file used in TLS configuration\n");
    printf("\n");
    printf("Notes:\n"
           "- If the public IP and port is not set, the transport will work in LAN mode,\n"
           "  where all the local IP address(es) will be used for the communication.\n"
           "- If the --port argument is not provided, the node will be unreachable and\n"
           "  will be able to communicate ONLY with reachable (server) peers\n"
```

```
            "- To use TLS configuration, you need to specify at least BOTH\n"
            "  --tls-cert and --tls-chain arguments\n");
}
#define ENSURE_ONE_MORE_ARGUMENT(argc, argv, i, arg_name)                    \
    if (i+1 >= argc) {                                                       \
        usage(appName);                                                      \
        fprintf(stderr, "! Error: missing value for %s argument\n",          \
                        arg_name);                                           \
        return EXIT_FAILURE;                                                 \
    }
/* Returns RTI_FALSE if failure, RTI_TRUE if success */
static RTIBool parseCmdline(struct CmdLineArgs *argOut, int argc, const char **argv) {
    int i;
    const char * const appName = argv[0];
    argOut->domain_id = 0;
    argOut->sample_count = 0;
    argOut->port = 0;
    argOut->addr = NULL;
    argOut->verbose = 0;
    argOut->asyncSend = RTI_FALSE;
    argOut->debug = RTI_FALSE;
    argOut->tls = RTI_FALSE;
    argOut->tlsCert = NULL;
    argOut->tlsChain = NULL;
    for (i = 1; i < argc; ++i) {
        if (!RTI_STRNCMP(argv[i], "-h", 2) ||
            !RTI_STRNCMP(argv[i], "--help", 6)) {
            usage(appName);
            return RTI_FALSE;
        }
        if (!RTI_STRNCMP(argv[i], "-v", 2) ||
            !RTI_STRNCMP(argv[i], "--verbose", 9)) {
            ++argOut->verbose;
            continue;
        }
        if (!RTI_STRNCMP(argv[i], "--async", 7)) {
            argOut->asyncSend = RTI_TRUE;
            continue;
        }
        if (!RTI_STRNCMP(argv[i], "--debug", 7)) {
            argOut->debug = RTI_TRUE;
            continue;
        }
        if (!RTI_STRNCMP(argv[i], "-d", 2) ||
            !RTI_STRNCMP(argv[i], "--domain", 8)) {
            char *ptr;
            ENSURE_ONE_MORE_ARGUMENT(argc, argv, i, "--domain")
            argOut->domain_id = strtol(argv[++i], &ptr, 10);
            if (*ptr != '\0') {
                usage(appName);
                fprintf(stderr, "! Value of --domain argument is not "
                        "a number: %s\n", argv[i]);
                return RTI_FALSE;
            }
            continue;
        }
        if (!RTI_STRNCMP(argv[i], "-s", 2) ||
            !RTI_STRNCMP(argv[i], "--sampleCount", 13)) {
            char *ptr;
            ENSURE_ONE_MORE_ARGUMENT(argc, argv, i, "--sampleCount")
            argOut->sample_count = strtol(argv[++i], &ptr, 10);
            if (*ptr != '\0') {
                usage(appName);
                fprintf(stderr, "! Value of --sampleCount argument is not "
                        "a number: %s\n", argv[i]);
                return RTI_FALSE;
            }
            continue;
        }
        if (!RTI_STRNCMP(argv[i], "-a", 2) ||
            !RTI_STRNCMP(argv[i], "--address", 9)) {
            ENSURE_ONE_MORE_ARGUMENT(argc, argv, i, "--address")
            argOut->addr = argv[++i];
            continue;
        }
        if (!RTI_STRNCMP(argv[i], "-p", 2) ||
            !RTI_STRNCMP(argv[i], "--port", 6)) {
            char *ptr;
            ENSURE_ONE_MORE_ARGUMENT(argc, argv, i, "--port")
            argOut->port = strtol(argv[++i], &ptr, 10);
            if (*ptr != '\0') {
```

```
                usage(appName);
                fprintf(stderr, "! Value of --port argument is not "
                        "a number: %s\n", argv[i]);
                return RTI_FALSE;
            }
            if (argOut->port < 0 || argOut->port > 65535) {
                usage(appName);
                fprintf(stderr, "! Invalid server port number: %d\n",
                        argOut->port);
                return RTI_FALSE;
            }
            continue;
        }
        if (!RTI_STRNCMP(argv[i], "--tls-cert", 10)) {
            ENSURE_ONE_MORE_ARGUMENT(argc, argv, i, "--tls-cert")
            argOut->tls = RTI_TRUE;
            argOut->tlsCert = argv[++i];
            continue;
        }
        if (!RTI_STRNCMP(argv[i], "--tls-chain", 11)) {
            ENSURE_ONE_MORE_ARGUMENT(argc, argv, i, "--tls-chain")
            argOut->tls = RTI_TRUE;
            argOut->tlsChain = argv[++i];
            continue;
        }
        usage(appName);
        fprintf(stderr, "! Unknown argument %s\n", argv[i]);
        return RTI_FALSE;
    } /* foreach argument */
    printf("Running with the following arguments:\n");
    printf("    Domain ID........ : %d\n", argOut->domain_id);
    printf("    Sample count..... : %d\n", argOut->sample_count);
    printf("    Public address... : %s\n",
           (argOut->addr ? argOut->addr : "<not set>"));
    printf("    Server port...... : %d\n", argOut->port);
    printf("    Verbosity........ : %d\n", argOut->verbose);
    printf("    AsyncSend........ : %s\n", (argOut->asyncSend ? "ENABLED" : "disabled"));
    printf("    Use TLS.......... : %s\n", (argOut->tls ? "YES" : "NO"));
    printf("    TLS CA Authority. : %s\n", (argOut->tlsCert ? argOut->tlsCert : "<not set>"));
    printf("    TLS Cert chain... : %s\n", (argOut->tlsChain ? argOut->tlsChain : "<not set>"));
    return RTI_TRUE;
}
DDS_ReturnCode_t configureTcpTransport(
        const struct CmdLineArgs * const args,
        struct DDS_DomainParticipantQos * const part_qos)
{
    DDS_ReturnCode_t rc;
#ifdef RTI_STATIC
    /* create function ptr */
    rc = DDS_PropertyQosPolicyHelper_assert_pointer_property(
            &part_qos->property,
            "dds.transport.tcp.tcp1.create_function_ptr",
            NDDS_Transport_TCPv4_create);
    if (rc != DDS_RETCODE_OK) {
        fprintf(stderr,
                "! unable to set property "
                "'dds.transport.tcp.tcp1.create_function_ptr': %d\n",
                rc);
        return DDS_RETCODE_ERROR;
    }
#else
    /* library */
    rc = DDS_PropertyQosPolicyHelper_assert_property(
            &part_qos->property,
            "dds.transport.tcp.tcp1.library",
            (args->debug ? TRANSPORT_LIB_NAME_DBG : TRANSPORT_LIB_NAME),
            DDS_BOOLEAN_FALSE);
    if (rc != DDS_RETCODE_OK) {
        fprintf(stderr,
                "! unable to set property 'dds.transport.tcp.tcp1.library': "
                "%d\n",
                rc);
        return DDS_RETCODE_ERROR;
    }
    rc = DDS_PropertyQosPolicyHelper_assert_property(
            &part_qos->property,
            "dds.transport.tcp.tcp1.create_function",
            "NDDS_Transport_TCPv4_create",
            DDS_BOOLEAN_FALSE);
    if (rc != DDS_RETCODE_OK) {
        fprintf(stderr,
```

```
                "! unable to set property "
                "'dds.transport.tcp.tcp1.create_function': %d\n",
                rc);
        return DDS_RETCODE_ERROR;
    }
#endif
    if (args->tls) {
#if (defined(RTI_STATIC) && defined(USE_TLS))
        rc = DDS_PropertyQosPolicyHelper_assert_pointer_property(
                &part_qos->property,
                "dds.transport.tcp.tcp1.tls_create_function_ptr",
                RTITLS_ConnectionEndpointFactoryTLSv4_create);
        if (rc != DDS_RETCODE_OK) {
            fprintf(stderr,
                    "! unable to set property "
                    "'dds.transport.tcp.tcp1.tls_create_function_ptr': %d\n",
                    rc);
            return DDS_RETCODE_ERROR;
        }
        rc = DDS_PropertyQosPolicyHelper_assert_pointer_property(
                &part_qos->property,
                "dds.transport.tcp.tcp1.tls_delete_function_ptr",
                RTITLS_ConnectionEndpointFactoryTLSv4_delete);
        if (rc != DDS_RETCODE_OK) {
            fprintf(stderr,
                    "! unable to set property "
                    "'dds.transport.tcp.tcp1.tls_delete_function_ptr': %d\n",
                    rc);
            return DDS_RETCODE_ERROR;
        }
#endif
        if (args->tlsCert == NULL) {
            fprintf(stderr, "! missing --tls-cert argument\n");
            return DDS_RETCODE_ERROR;
        }
        if (args->tlsChain == NULL) {
            fprintf(stderr, "! missing --tls-chain argument\n");
            return DDS_RETCODE_ERROR;
        }
        rc = DDS_PropertyQosPolicyHelper_assert_property(
                &part_qos->property,
                "dds.transport.tcp.tcp1.tls.verify.ca_file",
                args->tlsCert,
                DDS_BOOLEAN_FALSE);
        if (rc != DDS_RETCODE_OK) {
            fprintf(stderr,
                    "! unable to set property "
                    "'dds.transport.tcp.tcp1.tls.verify.ca_file': %d\n",
                    rc);
            return DDS_RETCODE_ERROR;
        }
        rc = DDS_PropertyQosPolicyHelper_assert_property(
                &part_qos->property,
                "dds.transport.tcp.tcp1.tls.identity.certificate_chain_file",
                args->tlsChain,
                DDS_BOOLEAN_FALSE);
        if (rc != DDS_RETCODE_OK) {
            fprintf(stderr,
                    "! unable to set property "
                    "'dds.transport.tcp.tcp1.tls.verify.certificate_chain_file'"
                    ": %d\n",
                    rc);
            return DDS_RETCODE_ERROR;
        }
    }
    if (args->port) {
        /* Node is REACHABLE. Internal server is LISTENING */
        char portstr[20];
        RTIOsapiUtility_snprintf(portstr, 20, "%d", args->port);
        rc = DDS_PropertyQosPolicyHelper_assert_property(
                &part_qos->property,
                "dds.transport.tcp.tcp1.server_bind_port",
                portstr,
                DDS_BOOLEAN_FALSE);
        if (rc != DDS_RETCODE_OK) {
            fprintf(stderr,
                    "! unable to set property "
                    "'dds.transport.tcp.tcp1.server_bind_port': %d\n",
                    rc);
            return DDS_RETCODE_ERROR;
        }
```

```
/* Deprecated
rc = DDS_PropertyQosPolicyHelper_assert_property(
        &part_qos->property,
        "dds.transport.tcp.tcp1.is_reachable",
        "1",
        DDS_BOOLEAN_FALSE);
if (rc != DDS_RETCODE_OK) {
    fprintf(stderr,
            "! unable to set property "
            "'dds.transport.tcp.tcp1.is_reachable': %d\n",
            rc);
    return DDS_RETCODE_ERROR;
}
*/
if (args->addr != NULL) {
    /* transport is in wan configuration */
    if (args->tls) {
        rc = DDS_PropertyQosPolicyHelper_assert_property(
                &part_qos->property,
                "dds.transport.tcp.tcp1.parent.classid",
                "NDDS_TRANSPORT_CLASSID_TLSV4_WAN",
                DDS_BOOLEAN_FALSE);
        if (rc != DDS_RETCODE_OK) {
            fprintf(stderr,
                    "! unable to set property "
                    "'dds.transport.tcp.tcp1.parent.classid': %d\n",
                    rc);
            return DDS_RETCODE_ERROR;
        }
    } else {
        rc = DDS_PropertyQosPolicyHelper_assert_property(
                &part_qos->property,
                "dds.transport.tcp.tcp1.parent.classid",
                "NDDS_TRANSPORT_CLASSID_TCPV4_WAN",
                DDS_BOOLEAN_FALSE);
        if (rc != DDS_RETCODE_OK) {
            fprintf(stderr,
                    "! unable to set property "
                    "'dds.transport.tcp.tcp1.parent.classid': %d\n",
                    rc);
            return DDS_RETCODE_ERROR;
        }
    }
    rc = DDS_PropertyQosPolicyHelper_assert_property(
            &part_qos->property,
            "dds.transport.tcp.tcp1.public_address",
            args->addr,
            DDS_BOOLEAN_FALSE);
    if (rc != DDS_RETCODE_OK) {
        fprintf(stderr,
                "! unable to set property "
                "'dds.transport.tcp.tcp1.public_address': %d\n",
                rc);
        return DDS_RETCODE_ERROR;
    }
    /*
    retcode = DDS_PropertyQosPolicyHelper_assert_property(
            &domainparticipant_qos.property,
            "dds.transport.TCP.tcp1.public_address.network_ordered_value",
            "0,0,0,0,0,0,0,0,255,255,28,232,192,168,93,100",
            DDS_BOOLEAN_FALSE);
    */
} else {
    /* transport is in LAN configuration */
    if (args->tls) {
        rc = DDS_PropertyQosPolicyHelper_assert_property(
                &part_qos->property,
                "dds.transport.tcp.tcp1.parent.classid",
                "NDDS_TRANSPORT_CLASSID_TLSV4_LAN",
                DDS_BOOLEAN_FALSE);
        if (rc != DDS_RETCODE_OK) {
            fprintf(stderr,
                    "! unable to set property "
                    "'dds.transport.tcp.tcp1.parent.classid': %d\n",
                    rc);
            return DDS_RETCODE_ERROR;
        }
    } else {
        rc = DDS_PropertyQosPolicyHelper_assert_property(
                &part_qos->property,
                "dds.transport.tcp.tcp1.parent.classid",
```

```
                    "NDDS_TRANSPORT_CLASSID_TCPV4_LAN",
                    DDS_BOOLEAN_FALSE);
            if (rc != DDS_RETCODE_OK) {
                fprintf(stderr,
                        "! unable to set property "
                        "'dds.transport.tcp.tcp1.parent.classid': %d\n",
                        rc);
                return DDS_RETCODE_ERROR;
            }
        }
    }
} else {
    /* node is NOT REACHABLE */
    if (args->addr != NULL) {
        printf("WARNING: You specified public address (--address), but you configured this node\n"
               "         to be NOT REACHABLE (you didn't specify the --port argument.\n"
               "         The public address will be ignored.\n");
    }
    /* Always use WAN configuration in asymmetric scenario */
    if (args->tls) {
        rc = DDS_PropertyQosPolicyHelper_assert_property(
                &part_qos->property,
                "dds.transport.tcp.tcp1.parent.classid",
                "NDDS_TRANSPORT_CLASSID_TLSV4_WAN",
                DDS_BOOLEAN_FALSE);
        if (rc != DDS_RETCODE_OK) {
            fprintf(stderr,
                    "! unable to set property "
                    "'dds.transport.tcp.tcp1.parent.classid': %d\n",
                    rc);
            return DDS_RETCODE_ERROR;
        }
    } else {
        rc = DDS_PropertyQosPolicyHelper_assert_property(
                &part_qos->property,
                "dds.transport.tcp.tcp1.parent.classid",
                "NDDS_TRANSPORT_CLASSID_TCPV4_WAN",
                DDS_BOOLEAN_FALSE);
        if (rc != DDS_RETCODE_OK) {
            fprintf(stderr,
                    "! unable to set property "
                    "'dds.transport.tcp.tcp1.parent.classid': %d\n",
                    rc);
            return DDS_RETCODE_ERROR;
        }
    }
    rc = DDS_PropertyQosPolicyHelper_assert_property(
            &part_qos->property,
            "dds.transport.tcp.tcp1.server_bind_port",
            "0",
            DDS_BOOLEAN_FALSE);
    if (rc != DDS_RETCODE_OK) {
        fprintf(stderr,
                "! unable to set property "
                "'dds.transport.tcp.tcp1.server_bind_port': %d\n",
                rc);
        return DDS_RETCODE_ERROR;
    }
}
/* Enable debugging logging */
if (args->verbose > 0) {
    /* Set transport verbosity */
    rc = DDS_PropertyQosPolicyHelper_assert_property(
            &part_qos->property,
            "dds.transport.tcp.tcp1.logging_verbosity_bitmap",
            "255",
            DDS_BOOLEAN_FALSE);
    if (rc != DDS_RETCODE_OK) {
        fprintf(stderr,
                "! unable to set property "
                "'dds.transport.tcp.tcp1.logging_verbosity_bitmap': %d\n",
                rc);
        return DDS_RETCODE_ERROR;
    }
}
/* Use async send */
rc = DDS_PropertyQosPolicyHelper_assert_property(
        &part_qos->property,
        "dds.transport.tcp.tcp1.force_asynchronous_send",
        (args->asyncSend ? "1" : "0"),
        DDS_BOOLEAN_FALSE);
```

```
    if (rc != DDS_RETCODE_OK) {
        fprintf(stderr,
                "! unable to set property "
                "'dds.transport.tcp.tcp1.force_asynchronous_send': %d\n",
                rc);
        return DDS_RETCODE_ERROR;
    }
    return DDS_RETCODE_OK;
}
```

5.12 HelloWorld.c

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorld.h]

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.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 HelloWorld_1436885484_h
#define HelloWorld_1436885484_h
#ifndef NDDS_STANDALONE_TYPE
#ifndef ndds_c_h
#include "ndds/ndds_c.h"
#endif
#include "cdr/cdr_typeCode.h"
#else
#include "ndds_standalone_type.h"
#endif
#ifdef __cplusplus
extern "C" {
    #endif
    extern const char *HelloWorldTYPENAME;
    typedef struct HelloWorld
    {
        DDS_Char * msg;
    } HelloWorld ;
    #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
    #ifndef NDDS_STANDALONE_TYPE
    NDDSUSERDllExport DDS_TypeCode * HelloWorld_get_typecode(void); /* Type code */
    NDDSUSERDllExport RTIXCdrTypePlugin *HelloWorld_get_type_plugin_info(void);
    NDDSUSERDllExport RTIXCdrSampleAccessInfo *HelloWorld_get_sample_access_info(void);
    #endif
    DDS_SEQUENCE(HelloWorldSeq, HelloWorld);
    NDDSUSERDllExport
    RTIBool HelloWorld_initialize(
        HelloWorld* self);
    NDDSUSERDllExport
    RTIBool HelloWorld_initialize_ex(
        HelloWorld* self,RTIBool allocatePointers,RTIBool allocateMemory);
    NDDSUSERDllExport
    RTIBool HelloWorld_initialize_w_params(
        HelloWorld* self,
        const struct DDS_TypeAllocationParams_t * allocParams);
    NDDSUSERDllExport
    RTIBool HelloWorld_finalize_w_return(
        HelloWorld* self);
    NDDSUSERDllExport
    void HelloWorld_finalize(
        HelloWorld* self);
    NDDSUSERDllExport
    void HelloWorld_finalize_ex(
        HelloWorld* self,RTIBool deletePointers);
    NDDSUSERDllExport
    void HelloWorld_finalize_w_params(
```

```
        HelloWorld* self,
        const struct DDS_TypeDeallocationParams_t * deallocParams);
    NDDSUSERDllExport
    void HelloWorld_finalize_optional_members(
        HelloWorld* self, RTIBool deletePointers);
    NDDSUSERDllExport
    RTIBool HelloWorld_copy(
        HelloWorld* dst,
        const HelloWorld* src);
    #if defined(NDDS_USER_DLL_EXPORT) || defined(NDDS_USER_SYMBOL_EXPORT)
    #undef NDDSUSERDllExport
    #define NDDSUSERDllExport
    #endif
    #ifdef __cplusplus
}
#endif
#endif /* HelloWorld */
```

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorld.c]

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.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 NDDS_STANDALONE_TYPE
#ifndef ndds_c_h
#include "ndds/ndds_c.h"
#endif
#ifndef dds_c_log_infrastructure_h
#include "dds_c/dds_c_infrastructure_impl.h"
#endif
#ifndef cdr_type_h
#include "cdr/cdr_type.h"
#endif
#include "osapi/osapi_atomic.h"
#else
#include "ndds_standalone_type.h"
#endif
#include "HelloWorld.h"
#ifndef NDDS_STANDALONE_TYPE
#include "HelloWorldPlugin.h"
#endif
/* ========================================================================= */
const char *HelloWorldTYPENAME = "HelloWorld";
#ifndef NDDS_STANDALONE_TYPE
DDS_TypeCode * HelloWorld_get_typecode(void)
{
    static RTI_ATOMIC(RTIBool) is_initialized;
    static DDS_TypeCode HelloWorld_g_tc_msg_string = DDS_INITIALIZE_STRING_TYPECODE((128L));
    static DDS_TypeCode_Member HelloWorld_g_tc_members[1]=
    {
        {
            (char *)"msg",/* Member name */
            {
                0,/* Representation ID */
                DDS_BOOLEAN_FALSE,/* Is a pointer? */
                -1, /* Bitfield bits */
                NULL/* Member type code is assigned later */
            },
            0, /* Ignored */
            0, /* Ignored */
            0, /* Ignored */
            NULL, /* Ignored */
            RTI_CDR_REQUIRED_MEMBER, /* Is a key? */
            DDS_PUBLIC_MEMBER,/* Member visibility */
            RTICdrTypeCodeAnnotations_INITIALIZER
        }
    };
    static DDS_TypeCode HelloWorld_g_tc =
    {{
            DDS_TK_STRUCT, /* Kind */
            DDS_BOOLEAN_FALSE, /* Ignored */
            -1, /*Ignored*/
            (char *)"HelloWorld", /* Name */
            NULL, /* Ignored */
            0, /* Ignored */
```

```
            0, /* Ignored */
            NULL, /* Ignored */
            1, /* Number of members */
            HelloWorld_g_tc_members, /* Members */
            DDS_VM_NONE, /* Ignored */
            RTICdrTypeCodeAnnotations_INITIALIZER,
            DDS_BOOLEAN_TRUE, /* _isCopyable */
            NULL, /* _sampleAccessInfo: assigned later */
            NULL /* _typePlugin: assigned later */
        }}; /* Type code for HelloWorld*/
    if (RTIOsapiAtomic_load32(&is_initialized, RTI_OSAPI_ATOMIC_MEMORY_ORDER_ACQUIRE)) {
        return &HelloWorld_g_tc;
    }
    HelloWorld_g_tc._data._annotations._allowedDataRepresentationMask = 5;
    HelloWorld_g_tc_members[0]._representation._typeCode =  (RTICdrTypeCode *)&HelloWorld_g_tc_msg_string;
    /* Initialize the values for member annotations. */
    HelloWorld_g_tc_members[0]._annotations._defaultValue._d = RTI_XCDR_TK_STRING;
    HelloWorld_g_tc_members[0]._annotations._defaultValue._u.string_value = (DDS_Char *) "";
    HelloWorld_g_tc._data._sampleAccessInfo =
    HelloWorld_get_sample_access_info();
    HelloWorld_g_tc._data._typePlugin =
    HelloWorld_get_type_plugin_info();
    RTIOsapiAtomic_store32(
        &is_initialized,
        RTI_TRUE,
        RTI_OSAPI_ATOMIC_MEMORY_ORDER_RELEASE);
    return &HelloWorld_g_tc;
}
RTIXCdrSampleAccessInfo *HelloWorld_get_sample_access_info()
{
    static RTI_ATOMIC(RTIBool) is_initialized;
    static RTIXCdrMemberAccessInfo HelloWorld_g_memberAccessInfos[1] =
    {RTIXCdrMemberAccessInfo_INITIALIZER};
    static RTIXCdrSampleAccessInfo HelloWorld_g_sampleAccessInfo =
    RTIXCdrSampleAccessInfo_INITIALIZER;
    if (RTIOsapiAtomic_load32(
        &is_initialized,
        RTI_OSAPI_ATOMIC_MEMORY_ORDER_ACQUIRE)) {
        return (RTIXCdrSampleAccessInfo*) &HelloWorld_g_sampleAccessInfo;
    }
    HelloWorld_g_memberAccessInfos[0].bindingMemberValueOffset[0] =
    offsetof(struct HelloWorld, msg);
    HelloWorld_g_sampleAccessInfo.memberAccessInfos =
    HelloWorld_g_memberAccessInfos;
    {
        size_t candidateTypeSize = sizeof(HelloWorld);
        if (candidateTypeSize > RTIXCdrLong_MAX) {
            HelloWorld_g_sampleAccessInfo.typeSize[0] =
            RTIXCdrLong_MAX;
        } else {
            HelloWorld_g_sampleAccessInfo.typeSize[0] =
            (RTIXCdrUnsignedLong) candidateTypeSize;
        }
    }
    HelloWorld_g_sampleAccessInfo.languageBinding =
    RTI_XCDR_TYPE_BINDING_C ;
    RTIOsapiAtomic_store32(
        &is_initialized,
        RTI_TRUE,
        RTI_OSAPI_ATOMIC_MEMORY_ORDER_RELEASE);
    return (RTIXCdrSampleAccessInfo*) &HelloWorld_g_sampleAccessInfo;
}
RTIXCdrTypePlugin *HelloWorld_get_type_plugin_info()
{
    static RTIXCdrTypePlugin HelloWorld_g_typePlugin =
    {
        NULL, /* serialize */
        NULL, /* serialize_key */
        NULL, /* deserialize_sample */
        NULL, /* deserialize_key_sample */
        NULL, /* skip */
        NULL, /* get_serialized_sample_size */
        NULL, /* get_serialized_sample_max_size_ex */
        NULL, /* get_serialized_key_max_size_ex */
        NULL, /* get_serialized_sample_min_size */
        NULL, /* serialized_sample_to_key */
        (RTIXCdrTypePluginInitializeSampleFunction)
        HelloWorld_initialize_ex,
        NULL,
        (RTIXCdrTypePluginFinalizeSampleFunction)
        HelloWorld_finalize_w_return,
```

```
        NULL,
        NULL
    };
    return &HelloWorld_g_typePlugin;
}
#endif
RTIBool HelloWorld_initialize(
    HelloWorld* sample)
{
    return HelloWorld_initialize_ex(
        sample,
        RTI_TRUE,
        RTI_TRUE);
}
RTIBool HelloWorld_initialize_w_params(
    HelloWorld *sample,
    const struct DDS_TypeAllocationParams_t *allocParams)
{
    if (sample == NULL) {
        return RTI_FALSE;
    }
    if (allocParams == NULL) {
        return RTI_FALSE;
    }
    if (allocParams->allocate_memory) {
        sample->msg = DDS_String_alloc((128L));
        if (sample->msg != NULL) {
            RTIOsapiUtility_unusedReturnValue(
                RTICdrType_copyStringEx(
                    &sample->msg,
                    "",
                    (128L),
                    RTI_FALSE),
                    RTIBool);
        }
        if (sample->msg == NULL) {
            return RTI_FALSE;
        }
    } else {
        if (sample->msg != NULL) {
            RTIOsapiUtility_unusedReturnValue(
                RTICdrType_copyStringEx(
                    &sample->msg,
                    "",
                    (128L),
                    RTI_FALSE),
                    RTIBool);
            if (sample->msg == NULL) {
                return RTI_FALSE;
            }
        }
    }
    return RTI_TRUE;
}
RTIBool HelloWorld_initialize_ex(
    HelloWorld *sample,
    RTIBool allocatePointers,
    RTIBool allocateMemory)
{
    struct DDS_TypeAllocationParams_t allocParams =
    DDS_TYPE_ALLOCATION_PARAMS_DEFAULT;
    allocParams.allocate_pointers =  (DDS_Boolean)allocatePointers;
    allocParams.allocate_memory = (DDS_Boolean)allocateMemory;
    return HelloWorld_initialize_w_params(
        sample,
        &allocParams);
}
RTIBool HelloWorld_finalize_w_return(
    HelloWorld* sample)
{
    HelloWorld_finalize_ex(sample, RTI_TRUE);
    return RTI_TRUE;
}
void HelloWorld_finalize(
    HelloWorld* sample)
{
    HelloWorld_finalize_ex(
        sample,
        RTI_TRUE);
}
void HelloWorld_finalize_ex(
```

```
    HelloWorld *sample,
    RTIBool deletePointers)
{
    struct DDS_TypeDeallocationParams_t deallocParams =
    DDS_TYPE_DEALLOCATION_PARAMS_DEFAULT;
    if (sample==NULL) {
        return;
    }
    deallocParams.delete_pointers = (DDS_Boolean)deletePointers;
    HelloWorld_finalize_w_params(
        sample,
        &deallocParams);
}
void HelloWorld_finalize_w_params(
    HelloWorld *sample,
    const struct DDS_TypeDeallocationParams_t *deallocParams)
{
    if (sample==NULL) {
        return;
    }
    if (deallocParams == NULL) {
        return;
    }
    if (sample->msg != NULL) {
        DDS_String_free(sample->msg);
        sample->msg=NULL;
    }
}
void HelloWorld_finalize_optional_members(
    HelloWorld* sample, RTIBool deletePointers)
{
    struct DDS_TypeDeallocationParams_t deallocParamsTmp =
    DDS_TYPE_DEALLOCATION_PARAMS_DEFAULT;
    struct DDS_TypeDeallocationParams_t * deallocParams =
    &deallocParamsTmp;
    if (sample==NULL) {
        return;
    }
    RTIOsapiUtility_unusedParameter(deallocParams);
    deallocParamsTmp.delete_pointers = (DDS_Boolean)deletePointers;
    deallocParamsTmp.delete_optional_members = DDS_BOOLEAN_TRUE;
}
RTIBool HelloWorld_copy(
    HelloWorld* dst,
    const HelloWorld* src)
{
    if (dst == NULL || src == NULL) {
        return RTI_FALSE;
    }
    if (!RTICdrType_copyStringEx (
        &dst->msg
        ,
        src->msg,
        (128L) + 1,
        RTI_FALSE)){
        return RTI_FALSE;
    }
    return RTI_TRUE;
}
#define T HelloWorld
#define TSeq HelloWorldSeq
#define T_initialize_w_params HelloWorld_initialize_w_params
#define T_finalize_w_params   HelloWorld_finalize_w_params
#define T_copy       HelloWorld_copy
#ifndef NDDS_STANDALONE_TYPE
#include "dds_c/generic/dds_c_sequence_TSeq.gen"
#else
#include "dds_c_sequence_TSeq.gen"
#endif
#undef T_copy
#undef T_finalize_w_params
#undef T_initialize_w_params
#undef TSeq
#undef T
```

5.13 HelloWorld_publisher.c

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorld_publisher.c]

```
/* HelloWorld_publisher.c

   A publication of data of type HelloWorld

   This file is derived from code automatically generated by the rtiddsgen
   command:

   rtiddsgen -language C -example <arch> HelloWorld.idl

   Example publication of type HelloWorld automatically generated by
   'rtiddsgen'. To test them follow these steps:

   (1) Compile this file and the example subscription.

   (2) Start the subscription on the same domain used for RTI Connext
       with the command
       objs/<arch>/HelloWorld_subscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Connext
       with the command
       objs/<arch>/HelloWorld_publisher <domain_id> <sample_count>

   (4) [Optional] Specify the list of discovery initial peers and
       multicast receive addresses via an environment variable or a file
       (in the current working directory) called NDDS_DISCOVERY_PEERS.

   You can run any number of publishers and subscribers programs, and can
   add and remove them dynamically from the domain.


   Example:

       To run the example application on domain <domain_id>:

       On Unix:

       objs/<arch>/HelloWorld_publisher <arguments>
       objs/<arch>/HelloWorld_subscriber <arguments>

       On Windows:

       objs\<arch>\HelloWorld_publisher <arguments>
       objs\<arch>\HelloWorld_subscriber <arguments>

       Use the option '--help' to print a list of all
       the accepted command-line options.


modification history
------------ -------
*/
#include <stdio.h>
#include <stdlib.h>
#include "ndds/ndds_c.h"
#include "HelloWorld.h"
#include "HelloWorldSupport.h"
#include "HelloCommon.h"
int RTI_SNPRINTF (
   char *buffer,
   size_t count,
   const char *format, ...)
{
    int length;
    va_list ap;
    va_start(ap, format);
#ifdef WIN32
    length = _vsnprintf_s(buffer, count, count, format, ap);
#else
    length = vsnprintf(buffer, count, format, ap);
#endif
     va_end(ap);
     return length;
}
/* Delete all entities */
static int publisher_shutdown(
    DDS_DomainParticipant *participant)
```

```
{
    DDS_ReturnCode_t retcode;
    int status = 0;
    if (participant != NULL) {
        retcode = DDS_DomainParticipant_delete_contained_entities(participant);
        if (retcode != DDS_RETCODE_OK) {
            fprintf(stderr, "delete_contained_entities error %d\n", retcode);
            status = -1;
        }
        retcode = DDS_DomainParticipantFactory_delete_participant(
            DDS_TheParticipantFactory, participant);
        if (retcode != DDS_RETCODE_OK) {
            fprintf(stderr, "delete_participant error %d\n", retcode);
            status = -1;
        }
    }
    /* RTI Connext provides finalize_instance() method on
       domain participant factory for people who want to release memory used
       by the participant factory. Uncomment the following block of code for
       clean destruction of the singleton. */
/*
    retcode = DDS_DomainParticipantFactory_finalize_instance();
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "finalize_instance error %d\n", retcode);
        status = -1;
    }
*/
    return status;
}
static int publisher_main(const struct CmdLineArgs * const args)
{
    DDS_DomainParticipant *participant = NULL;
    DDS_Publisher *publisher = NULL;
    DDS_Topic *topic = NULL;
    DDS_DataWriter *writer = NULL;
    HelloWorldDataWriter *HelloWorld_writer = NULL;
    HelloWorld *instance = NULL;
    DDS_ReturnCode_t retcode;
    DDS_InstanceHandle_t instance_handle = DDS_HANDLE_NIL;
    struct DDS_DomainParticipantQos domain_participant_qos =
            DDS_DomainParticipantQos_INITIALIZER;
    const char *type_name = NULL;
    int count = 0;
    struct DDS_Duration_t send_period = {4,0};
    int result = -1;
    retcode = DDS_DomainParticipantFactory_get_default_participant_qos(
            DDS_TheParticipantFactory,
            &domain_participant_qos);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "get_default_participant_qos error\n");
        goto done;
    }
    /* Now we configure the TCP transport */
    retcode = configureTcpTransport(args, &domain_participant_qos);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "assert_property error\n");
        goto done;
    }
    participant = DDS_DomainParticipantFactory_create_participant(
            DDS_TheParticipantFactory,
            args->domain_id,
            &domain_participant_qos,
            NULL /* listener */,
            DDS_STATUS_MASK_NONE);
    if (participant == NULL) {
        fprintf(stderr, "create_participant error\n");
        goto done;
    }
    /*
     * To customize publisher QoS, use
     * the configuration file USER_QOS_PROFILES.xml
     */
    publisher = DDS_DomainParticipant_create_publisher(
            participant,
            &DDS_PUBLISHER_QOS_DEFAULT,
            NULL /* listener */,
            DDS_STATUS_MASK_NONE);
    if (publisher == NULL) {
        fprintf(stderr, "create_publisher error\n");
        goto done;
    }
```

```
    /* Register type before creating topic */
    type_name = HelloWorldTypeSupport_get_type_name();
    retcode = HelloWorldTypeSupport_register_type(participant, type_name);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "register_type error %d\n", retcode);
        goto done;
    }
    /*
     * To customize topic QoS, use
     * the configuration file USER_QOS_PROFILES.xml
     */
    topic = DDS_DomainParticipant_create_topic(
            participant,
            "Example HelloWorld",
            type_name,
            &DDS_TOPIC_QOS_DEFAULT,
            NULL /* listener */,
            DDS_STATUS_MASK_NONE);
    if (topic == NULL) {
        fprintf(stderr, "create_topic error\n");
        goto done;
    }
    /*
     * To customize data writer QoS, use
     * the configuration file USER_QOS_PROFILES.xml
     */
    writer = DDS_Publisher_create_datawriter(
            publisher,
            topic,
            &DDS_DATAWRITER_QOS_DEFAULT,
            NULL /* listener */,
            DDS_STATUS_MASK_NONE);
    if (writer == NULL) {
        fprintf(stderr, "create_datawriter error\n");
        goto done;
    }
    HelloWorld_writer = HelloWorldDataWriter_narrow(writer);
    if (HelloWorld_writer == NULL) {
        fprintf(stderr, "DataWriter narrow error\n");
        goto done;
    }
    /* Create data sample for writing */
    instance = HelloWorldTypeSupport_create_data_ex(DDS_BOOLEAN_TRUE);
    if (instance == NULL) {
        fprintf(stderr, "HelloWorldTypeSupport_create_data error\n");
        goto done;
    }
    /*
     * For a data type that has a key, if the same instance is going to be
     * written multiple times, initialize the key here
     * and register the keyed instance prior to writing
     */
    /*
        instance_handle = HelloWorldDataWriter_register_instance(
            HelloWorld_writer, instance);
    */
    /* Main loop */
    for (count=0; (args->sample_count == 0) || (count < args->sample_count); ++count) {
        printf("Writing HelloWorld, count %d\n", count);
        fflush(stdout);
        /* Modify the data to be written here */
        RTI_SNPRINTF(instance->msg, 128, "Hello, this is sample #%d", count);
        /* Write data */
        retcode = HelloWorldDataWriter_write(
                HelloWorld_writer,
                instance,
                &instance_handle);
        if (retcode != DDS_RETCODE_OK) {
            fprintf(stderr, "write error %d\n", retcode);
        }
        NDDS_Utility_sleep(&send_period);
    }
    result = 0;
done:
    /*
        retcode = HelloWorldDataWriter_unregister_instance(
            HelloWorld_writer, instance, &instance_handle);
        if (retcode != DDS_RETCODE_OK) {
            fprintf(stderr, "unregister instance error %d\n", retcode);
        }
    */
```

```
    /* Delete data sample */
    retcode = HelloWorldTypeSupport_delete_data_ex(instance, DDS_BOOLEAN_TRUE);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "HelloWorldTypeSupport_delete_data error %d\n", retcode);
        result = -1;
    }
    /* Cleanup and delete delete all entities */
    if (publisher_shutdown(participant) != 0) {
        fprintf(stderr, "publisher_shutdown error\n");
        result = -1;
    }
    /* Cleanup any resource allocated with Domain Participant QoS */
    retcode = DDS_DomainParticipantQos_finalize(&domain_participant_qos);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr,
                "DDS_DomainParticipantQos_finalize error %d\n",
                retcode);
        result = -1;
    }
    return result;
}
int main(int argc, const char **argv)
{
    struct CmdLineArgs args;
    if (!parseCmdline(&args, argc, argv)) {
        return 1;
    }
    if (args.verbose > 1) {
        /*
         NDDS_Config_Logger_set_verbosity_by_category(
            NDDS_Config_Logger_get_instance(),
            NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION,
            NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
        */
         NDDS_Config_Logger_set_verbosity(
            NDDS_Config_Logger_get_instance(),
            NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    }
    return publisher_main(&args);
}
```

5.14 HelloWorld_subscriber.c

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorld_subscriber.c]

```
/* HelloWorld_subscriber.c

   A subscription example

   This file is derived from code automatically generated by the rtiddsgen
   command:

   rtiddsgen -language C -example <arch> HelloWorld.idl

   Example subscription of type HelloWorld automatically generated by
   'rtiddsgen'. To test them, follow these steps:

   (1) Compile this file and the example publication.

   (2) Start the subscription on the same domain used for RTI Connext with the command
       objs/<arch>/HelloWorld_subscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Connext with the command
       objs/<arch>/HelloWorld_publisher <domain_id> <sample_count>

   (4) [Optional] Specify the list of discovery initial peers and
       multicast receive addresses via an environment variable or a file
       (in the current working directory) called NDDS_DISCOVERY_PEERS.

   You can run any number of publishers and subscribers programs, and can
   add and remove them dynamically from the domain.


   Example:

       To run the example application on domain <domain_id>:
```

```
       On UNIX systems:

       objs/<arch>/HelloWorld_publisher <arguments>
       objs/<arch>/HelloWorld_subscriber <arguments>

       On Windows systems:

       objs\<arch>\HelloWorld_publisher <arguments>
       objs\<arch>\HelloWorld_subscriber <arguments>

       Use the option '--help' to print a list of all
       the accepted command-line options.


modification history
------------ -------
*/
#include <stdio.h>
#include <stdlib.h>
#include "ndds/ndds_c.h"
#include "HelloWorld.h"
#include "HelloWorldSupport.h"
#include "HelloCommon.h"
void HelloWorldListener_on_requested_deadline_missed(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_RequestedDeadlineMissedStatus *status)
{
}
void HelloWorldListener_on_requested_incompatible_qos(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_RequestedIncompatibleQosStatus *status)
{
}
void HelloWorldListener_on_sample_rejected(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_SampleRejectedStatus *status)
{
}
void HelloWorldListener_on_liveliness_changed(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_LivelinessChangedStatus *status)
{
}
void HelloWorldListener_on_sample_lost(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_SampleLostStatus *status)
{
}
void HelloWorldListener_on_subscription_matched(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_SubscriptionMatchedStatus *status)
{
}
void HelloWorldListener_on_data_available(
    void* listener_data,
    DDS_DataReader* reader)
{
    HelloWorldDataReader *HelloWorld_reader = NULL;
    struct HelloWorldSeq data_seq = DDS_SEQUENCE_INITIALIZER;
    struct DDS_SampleInfoSeq info_seq = DDS_SEQUENCE_INITIALIZER;
    DDS_ReturnCode_t retcode;
    int i;
    HelloWorld_reader = HelloWorldDataReader_narrow(reader);
    if (HelloWorld_reader == NULL) {
        fprintf(stderr, "DataReader narrow error\n");
        return;
    }
    retcode = HelloWorldDataReader_take(
        HelloWorld_reader,
        &data_seq, &info_seq, DDS_LENGTH_UNLIMITED,
        DDS_ANY_SAMPLE_STATE, DDS_ANY_VIEW_STATE, DDS_ANY_INSTANCE_STATE);
    if (retcode == DDS_RETCODE_NO_DATA) {
        return;
    } else if (retcode != DDS_RETCODE_OK) {
```

```
        fprintf(stderr, "take error %d\n", retcode);
        return;
    }
    for (i = 0; i < HelloWorldSeq_get_length(&data_seq); ++i) {
        if (DDS_SampleInfoSeq_get_reference(&info_seq, i)->valid_data) {
            HelloWorldTypeSupport_print_data(
                HelloWorldSeq_get_reference(&data_seq, i));
            fflush(stdout);
        }
    }
    retcode = HelloWorldDataReader_return_loan(
        HelloWorld_reader,
        &data_seq, &info_seq);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "return loan error %d\n", retcode);
    }
}
/* Delete all entities */
static int subscriber_shutdown(
    DDS_DomainParticipant *participant)
{
    DDS_ReturnCode_t retcode;
    int status = 0;
    if (participant != NULL) {
        retcode = DDS_DomainParticipant_delete_contained_entities(participant);
        if (retcode != DDS_RETCODE_OK) {
            fprintf(stderr, "delete_contained_entities error %d\n", retcode);
            status = -1;
        }
        retcode = DDS_DomainParticipantFactory_delete_participant(
            DDS_TheParticipantFactory, participant);
        if (retcode != DDS_RETCODE_OK) {
            fprintf(stderr, "delete_participant error %d\n", retcode);
            status = -1;
        }
    }
    /* RTI Connext provides the finalize_instance() method on
       domain participant factory for users who want to release memory used
       by the participant factory. Uncomment the following block of code for
       clean destruction of the singleton. */
/*
    retcode = DDS_DomainParticipantFactory_finalize_instance();
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "finalize_instance error %d\n", retcode);
        status = -1;
    }
*/
    return status;
}
static int subscriber_main(const struct CmdLineArgs * const args)
{
    DDS_DomainParticipant *participant = NULL;
    DDS_Subscriber *subscriber = NULL;
    DDS_Topic *topic = NULL;
    struct DDS_DataReaderListener reader_listener =
        DDS_DataReaderListener_INITIALIZER;
    DDS_DataReader *reader = NULL;
    DDS_ReturnCode_t retcode;
    const char *type_name = NULL;
    int count = 0;
    struct DDS_Duration_t poll_period = {4,0};
    struct DDS_DomainParticipantQos domain_participant_qos =
            DDS_DomainParticipantQos_INITIALIZER;
    int result = -1;
    retcode = DDS_DomainParticipantFactory_get_default_participant_qos(
            DDS_TheParticipantFactory,
            &domain_participant_qos);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "get_default_participant_qos error\n");
        goto done;
    }
    /* Now we configure the TCP transport */
    retcode = configureTcpTransport(args, &domain_participant_qos);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "assert_property error\n");
        goto done;
    }
    participant = DDS_DomainParticipantFactory_create_participant(
            DDS_TheParticipantFactory,
            args->domain_id,
            &domain_participant_qos,
```

```
            NULL /* listener */,
            DDS_STATUS_MASK_NONE);
    if (participant == NULL) {
        fprintf(stderr, "create_participant error\n");
        goto done;
    }
    /*
     * To customize subscriber QoS, use
     * the configuration file USER_QOS_PROFILES.xml
     */
    subscriber = DDS_DomainParticipant_create_subscriber(
            participant,
            &DDS_SUBSCRIBER_QOS_DEFAULT,
            NULL /* listener */,
            DDS_STATUS_MASK_NONE);
    if (subscriber == NULL) {
        fprintf(stderr, "create_subscriber error\n");
        goto done;
    }
    /* Register the type before creating the topic */
    type_name = HelloWorldTypeSupport_get_type_name();
    retcode = HelloWorldTypeSupport_register_type(participant, type_name);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "register_type error %d\n", retcode);
        goto done;
    }
    /*
     * To customize topic QoS, use
     * the configuration file USER_QOS_PROFILES.xml
     */
    topic = DDS_DomainParticipant_create_topic(
            participant,
            "Example HelloWorld",
            type_name,
            &DDS_TOPIC_QOS_DEFAULT,
            NULL /* listener */,
            DDS_STATUS_MASK_NONE);
    if (topic == NULL) {
        fprintf(stderr, "create_topic error\n");
        goto done;
    }
    /* Set up a data reader listener */
    reader_listener.on_requested_deadline_missed  =
        HelloWorldListener_on_requested_deadline_missed;
    reader_listener.on_requested_incompatible_qos =
        HelloWorldListener_on_requested_incompatible_qos;
    reader_listener.on_sample_rejected =
        HelloWorldListener_on_sample_rejected;
    reader_listener.on_liveliness_changed =
        HelloWorldListener_on_liveliness_changed;
    reader_listener.on_sample_lost =
        HelloWorldListener_on_sample_lost;
    reader_listener.on_subscription_matched =
        HelloWorldListener_on_subscription_matched;
    reader_listener.on_data_available =
        HelloWorldListener_on_data_available;
    /*
     * To customize data reader QoS, use
     * the configuration file USER_QOS_PROFILES.xml
     */
    reader = DDS_Subscriber_create_datareader(
            subscriber,
            DDS_Topic_as_topicdescription(topic),
            &DDS_DATAREADER_QOS_DEFAULT,
            &reader_listener,
            DDS_STATUS_MASK_ALL);
    if (reader == NULL) {
        fprintf(stderr, "create_datareader error\n");
        goto done;
    }
    /* Main loop */
    for (count=0; (args->sample_count == 0) || (count < args->sample_count); ++count) {
        printf("HelloWorld subscriber sleeping for %d sec...\n",
                poll_period.sec);
        fflush(stdout);
        NDDS_Utility_sleep(&poll_period);
    }
    result = 0;
done:
    /* Cleanup any resource allocated with Domain Participant QoS */
    retcode = DDS_DomainParticipantQos_finalize(&domain_participant_qos);
```

```
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr,
                "DDS_DomainParticipantQos_finalize error %d\n",
                retcode);
        result = -1;
    }
    /* Cleanup and delete delete all entities */
    if (subscriber_shutdown(participant) != 0) {
        fprintf(stderr, "subscriber_shutdown error\n");
        result = -1;
    }
    return result;
}
int main(int argc, const char **argv)
{
    struct CmdLineArgs args;
    if (!parseCmdline(&args, argc, argv)) {
        return 1;
    }
    if (args.verbose > 1) {
        /*
         NDDS_Config_Logger_set_verbosity_by_category(
            NDDS_Config_Logger_get_instance(),
            NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION,
            NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
        */
         NDDS_Config_Logger_set_verbosity(
            NDDS_Config_Logger_get_instance(),
            NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    }
    return subscriber_main(&args);
}
```

5.15 HelloWorldPlugin.c

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorldPlugin.h]

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.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 HelloWorldPlugin_1436885484_h
#define HelloWorldPlugin_1436885484_h
#include "HelloWorld.h"
struct RTICdrStream;
#ifndef pres_typePlugin_h
#include "pres/pres_typePlugin.h"
#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
#ifdef __cplusplus
extern "C" {
    #endif
    #define HelloWorldPlugin_get_sample PRESTypePluginDefaultEndpointData_getSample
    #define HelloWorldPlugin_get_buffer PRESTypePluginDefaultEndpointData_getBuffer
    #define HelloWorldPlugin_return_buffer PRESTypePluginDefaultEndpointData_returnBuffer
    #define HelloWorldPlugin_create_sample PRESTypePluginDefaultEndpointData_createSample
    #define HelloWorldPlugin_destroy_sample PRESTypePluginDefaultEndpointData_deleteSample
    /* --------------------------------------------------------------------------------------
    Support functions:
    * -------------------------------------------------------------------------------------- */
    NDDSUSERDllExport extern HelloWorld*
    HelloWorldPluginSupport_create_data_w_params(
        const struct DDS_TypeAllocationParams_t * alloc_params);
    NDDSUSERDllExport extern HelloWorld*
```

```
HelloWorldPluginSupport_create_data_ex(RTIBool allocate_pointers);
NDDSUSERDllExport extern HelloWorld*
HelloWorldPluginSupport_create_data(void);
NDDSUSERDllExport extern RTIBool
HelloWorldPluginSupport_copy_data(
    HelloWorld *out,
    const HelloWorld *in);
NDDSUSERDllExport extern void
HelloWorldPluginSupport_destroy_data_w_params(
    HelloWorld *sample,
    const struct DDS_TypeDeallocationParams_t * dealloc_params);
NDDSUSERDllExport extern void
HelloWorldPluginSupport_destroy_data_ex(
    HelloWorld *sample,RTIBool deallocate_pointers);
NDDSUSERDllExport extern void
HelloWorldPluginSupport_destroy_data(
    HelloWorld *sample);
NDDSUSERDllExport extern void
HelloWorldPluginSupport_print_data(
    const HelloWorld *sample,
    const char *desc,
    unsigned int indent);
/* ----------------------------------------------------------------------------
Callback functions:
* ---------------------------------------------------------------------------- */
NDDSUSERDllExport extern PRESTypePluginParticipantData
HelloWorldPlugin_on_participant_attached(
    void *registration_data,
    const struct PRESTypePluginParticipantInfo *participant_info,
    RTIBool top_level_registration,
    void *container_plugin_context,
    RTICdrTypeCode *typeCode);
NDDSUSERDllExport extern void
HelloWorldPlugin_on_participant_detached(
    PRESTypePluginParticipantData participant_data);
NDDSUSERDllExport extern PRESTypePluginEndpointData
HelloWorldPlugin_on_endpoint_attached(
    PRESTypePluginParticipantData participant_data,
    const struct PRESTypePluginEndpointInfo *endpoint_info,
    RTIBool top_level_registration,
    void *container_plugin_context);
NDDSUSERDllExport extern void
HelloWorldPlugin_on_endpoint_detached(
    PRESTypePluginEndpointData endpoint_data);
NDDSUSERDllExport extern void
HelloWorldPlugin_return_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
    void *handle);
NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_copy_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *out,
    const HelloWorld *in);
/* ----------------------------------------------------------------------------
(De)Serialize functions:
* ---------------------------------------------------------------------------- */
NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_serialize_to_cdr_buffer(
    char * buffer,
    unsigned int * length,
    const HelloWorld *sample);
NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_serialize_to_cdr_buffer_ex(
    char *buffer,
    unsigned int *length,
    const HelloWorld *sample,
    DDS_DataRepresentationId_t representation);
NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_deserialize_from_cdr_buffer(
    HelloWorld *sample,
    const char * buffer,
    unsigned int length);
#if !defined (NDDS_STANDALONE_TYPE)
NDDSUSERDllExport extern DDS_ReturnCode_t
HelloWorldPlugin_data_to_string(
    const HelloWorld *sample,
    char *str,
    DDS_UnsignedLong *str_size,
    const struct DDS_PrintFormatProperty *property);
#endif
```

```
    NDDSUSERDllExport extern unsigned int
    HelloWorldPlugin_get_serialized_sample_max_size(
        PRESTypePluginEndpointData endpoint_data,
        RTIBool include_encapsulation,
        RTIEncapsulationId encapsulation_id,
        unsigned int current_alignment);
    /* ------------------------------------------------------------------------------------
    Key Management functions:
    * ------------------------------------------------------------------------------------ */
    NDDSUSERDllExport extern PRESTypePluginKeyKind
    HelloWorldPlugin_get_key_kind(void);
    NDDSUSERDllExport extern unsigned int
    HelloWorldPlugin_get_serialized_key_max_size(
        PRESTypePluginEndpointData endpoint_data,
        RTIBool include_encapsulation,
        RTIEncapsulationId encapsulation_id,
        unsigned int current_alignment);
    NDDSUSERDllExport extern unsigned int
    HelloWorldPlugin_get_serialized_key_max_size_for_keyhash(
        PRESTypePluginEndpointData endpoint_data,
        RTIEncapsulationId encapsulation_id,
        unsigned int current_alignment);
    NDDSUSERDllExport extern RTIBool
    HelloWorldPlugin_deserialize_key(
        PRESTypePluginEndpointData endpoint_data,
        HelloWorld ** sample,
        RTIBool * drop_sample,
        struct RTICdrStream *cdrStream,
        RTIBool deserialize_encapsulation,
        RTIBool deserialize_key,
        void *endpoint_plugin_qos);
    NDDSUSERDllExport extern
    struct RTIXCdrInterpreterPrograms * HelloWorldPlugin_get_programs(void);
    /* Plugin Functions */
    NDDSUSERDllExport extern struct PRESTypePlugin*
    HelloWorldPlugin_new(void);
    NDDSUSERDllExport extern void
    HelloWorldPlugin_delete(struct PRESTypePlugin *);
    #ifdef __cplusplus
}
#endif
#if defined(NDDS_USER_DLL_EXPORT) || defined(NDDS_USER_SYMBOL_EXPORT)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport
#endif
#endif /* HelloWorldPlugin_1436885484_h */
```

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorldPlugin.c]

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.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.
*/
#include <string.h>
#ifndef ndds_c_h
#include "ndds/ndds_c.h"
#endif
#ifndef cdr_type_h
#include "cdr/cdr_type.h"
#endif
#ifndef cdr_type_object_h
#include "cdr/cdr_typeObject.h"
#endif
#ifndef cdr_encapsulation_h
#include "cdr/cdr_encapsulation.h"
#endif
#ifndef cdr_stream_h
#include "cdr/cdr_stream.h"
#endif
#include "xcdr/xcdr_interpreter.h"
#include "xcdr/xcdr_stream.h"
#ifndef cdr_log_h
#include "cdr/cdr_log.h"
#endif
#ifndef pres_typePlugin_h
#include "pres/pres_typePlugin.h"
```

```
#endif
#include "dds_c/dds_c_typecode_impl.h"
#define RTI_CDR_CURRENT_SUBMODULE RTI_CDR_SUBMODULE_MASK_STREAM
#include "HelloWorldPlugin.h"
/* ------------------------------------------------------------------------
*  Type HelloWorld
* ------------------------------------------------------------------------ */
/* ----------------------------------------------------------------------------
Support functions:
* ------------------------------------------------------------------------ */
HelloWorld*
HelloWorldPluginSupport_create_data_w_params(
    const struct DDS_TypeAllocationParams_t * alloc_params)
{
    HelloWorld *sample = NULL;
    if (alloc_params == NULL) {
        return NULL;
    } else if(!alloc_params->allocate_memory) {
        RTICdrLog_exception(&RTI_CDR_LOG_TYPE_OBJECT_NOT_ASSIGNABLE_ss,
        "alloc_params->allocate_memory","false");
        return NULL;
    }
    RTIOsapiHeap_allocateStructure(&(sample),HelloWorld);
    if (sample == NULL) {
        return NULL;
    }
    if (!HelloWorld_initialize_w_params(sample,alloc_params)) {
        struct DDS_TypeDeallocationParams_t deallocParams =
        DDS_TYPE_DEALLOCATION_PARAMS_DEFAULT;
        deallocParams.delete_pointers = alloc_params->allocate_pointers;
        deallocParams.delete_optional_members = alloc_params->allocate_pointers;
        /* Coverity reports a possible uninit_use_in_call that will happen if the
        allocation fails. But if the allocation fails then sample == null and
        the method will return before reach this point.*/
        /* Coverity reports a possible overwrite_var on the members of the sample.
        It is a false positive since all the pointers are freed before assigning
        null to them. */
        /* coverity[uninit_use_in_call : FALSE] */
        /* coverity[overwrite_var : FALSE] */
        HelloWorld_finalize_w_params(sample, &deallocParams);
        /* Coverity reports a possible leaked_storage on the sample members when
        freeing sample. It is a false positive since all the members' memory
        is freed in the call "HelloWorld_finalize_ex" */
        /* coverity[leaked_storage : FALSE] */
        RTIOsapiHeap_freeStructure(sample);
        sample=NULL;
    }
    return sample;
}
HelloWorld *
HelloWorldPluginSupport_create_data_ex(RTIBool allocate_pointers)
{
    HelloWorld *sample = NULL;
    RTIOsapiHeap_allocateStructure(&(sample),HelloWorld);
    if(sample == NULL) {
        return NULL;
    }
    /* coverity[example_checked : FALSE] */
    if (!HelloWorld_initialize_ex(sample,allocate_pointers, RTI_TRUE)) {
        /* Coverity reports a possible uninit_use_in_call that will happen if the
        new fails. But if new fails then sample == null and the method will
        return before reach this point. */
        /* Coverity reports a possible overwrite_var on the members of the sample.
        It is a false positive since all the pointers are freed before assigning
        null to them. */
        /* coverity[uninit_use_in_call : FALSE] */
        /* coverity[overwrite_var : FALSE] */
        HelloWorld_finalize_ex(sample, RTI_TRUE);
        /* Coverity reports a possible leaked_storage on the sample members when
        freeing sample. It is a false positive since all the members' memory
        is freed in the call "HelloWorld_finalize_ex" */
        /* coverity[leaked_storage : FALSE] */
        RTIOsapiHeap_freeStructure(sample);
        sample=NULL;
    }
    return sample;
}
void *
HelloWorldPluginSupport_create_dataI(void)
{
    return HelloWorldPluginSupport_create_data_ex(RTI_TRUE);
```

```
}
HelloWorld *
HelloWorldPluginSupport_create_data(void)
{
    return (HelloWorld *) HelloWorldPluginSupport_create_dataI();
}
void
HelloWorldPluginSupport_destroy_data_w_params(
    HelloWorld *sample,
    const struct DDS_TypeDeallocationParams_t * dealloc_params) {
    HelloWorld_finalize_w_params(sample,dealloc_params);
    RTIOsapiHeap_freeStructure(sample);
}
void
HelloWorldPluginSupport_destroy_data_ex(
    HelloWorld *sample,RTIBool deallocate_pointers) {
    HelloWorld_finalize_ex(sample,deallocate_pointers);
    RTIOsapiHeap_freeStructure(sample);
}
void
HelloWorldPluginSupport_destroy_data(
    HelloWorld *sample) {
    HelloWorldPluginSupport_destroy_data_ex(sample,RTI_TRUE);
}
void
HelloWorldPluginSupport_destroy_dataI(
    void *sample)
{
    HelloWorldPluginSupport_destroy_data((HelloWorld *) sample);
}
RTIBool
HelloWorldPluginSupport_copy_data(
    HelloWorld *dst,
    const HelloWorld *src)
{
    return HelloWorld_copy(dst,(const HelloWorld*) src);
}
void
HelloWorldPluginSupport_print_data(
    const HelloWorld *sample,
    const char *desc,
    unsigned int indent_level)
{
    RTICdrType_printIndent(indent_level);
    if (desc != NULL) {
        RTILogParamString_printPlain("%s:\n", desc);
    } else {
        RTILogParamString_printPlain("\n");
    }
    if (sample == NULL) {
        RTILogParamString_printPlain("NULL\n");
        return;
    }
    if (sample->msg==NULL) {
        RTICdrType_printString(
            NULL,
            "msg",
            RTIOsapiUtility_uInt32Plus1(indent_level));
    } else {
        RTICdrType_printString(
            sample->msg,
            "msg",
            RTIOsapiUtility_uInt32Plus1(indent_level));
    }
}
/* ------------------------------------------------------------------------
Callback functions:
* ------------------------------------------------------------------------ */
PRESTypePluginParticipantData
HelloWorldPlugin_on_participant_attached(
    void *registration_data,
    const struct PRESTypePluginParticipantInfo *participant_info,
    RTIBool top_level_registration,
    void *container_plugin_context,
    RTICdrTypeCode *type_code)
{
    struct RTIXCdrInterpreterPrograms *programs = NULL;
    struct RTIXCdrInterpreterProgramsGenProperty programProperty =
    RTIXCdrInterpreterProgramsGenProperty_INITIALIZER;
    struct PRESTypePluginDefaultParticipantData *pd = NULL;
    RTIOsapiUtility_unusedParameter(registration_data);
```

```
    RTIOsapiUtility_unusedParameter(participant_info);
    RTIOsapiUtility_unusedParameter(top_level_registration);
    RTIOsapiUtility_unusedParameter(container_plugin_context);
    RTIOsapiUtility_unusedParameter(type_code);
    pd = (struct PRESTypePluginDefaultParticipantData *)
    PRESTypePluginDefaultParticipantData_new(participant_info);
    programProperty.generateV1Encapsulation = RTI_XCDR_TRUE;
    programProperty.generateV2Encapsulation = RTI_XCDR_TRUE;
    programProperty.resolveAlias = RTI_XCDR_TRUE;
    programProperty.inlineStruct = RTI_XCDR_TRUE;
    programProperty.optimizeEnum = RTI_XCDR_TRUE;
    programProperty.unboundedSize = RTIXCdrLong_MAX;
    programs = DDS_TypeCodeFactory_assert_programs_in_global_list(
        DDS_TypeCodeFactory_get_instance(),
        HelloWorld_get_typecode(),
        &programProperty,
        RTI_XCDR_PROGRAM_MASK_TYPEPLUGIN);
    if (programs == NULL) {
        PRESTypePluginDefaultParticipantData_delete(
            (PRESTypePluginParticipantData) pd);
        return NULL;
    }
    pd->programs = programs;
    return (PRESTypePluginParticipantData)pd;
}
void
HelloWorldPlugin_on_participant_detached(
    PRESTypePluginParticipantData participant_data)
{
    if (participant_data != NULL) {
        struct PRESTypePluginDefaultParticipantData *pd =
        (struct PRESTypePluginDefaultParticipantData *)participant_data;
        if (pd->programs != NULL) {
            DDS_TypeCodeFactory_remove_programs_from_global_list(
                DDS_TypeCodeFactory_get_instance(),
                pd->programs);
            pd->programs = NULL;
        }
        PRESTypePluginDefaultParticipantData_delete(participant_data);
    }
}
PRESTypePluginEndpointData
HelloWorldPlugin_on_endpoint_attached(
    PRESTypePluginParticipantData participant_data,
    const struct PRESTypePluginEndpointInfo *endpoint_info,
    RTIBool top_level_registration,
    void *containerPluginContext)
{
    PRESTypePluginEndpointData epd = NULL;
    unsigned int serializedSampleMaxSize = 0;
    RTIOsapiUtility_unusedParameter(top_level_registration);
    RTIOsapiUtility_unusedParameter(containerPluginContext);
    if (participant_data == NULL) {
        return NULL;
    }
    epd = PRESTypePluginDefaultEndpointData_new(
        participant_data,
        endpoint_info,
        HelloWorldPluginSupport_create_dataI,
        HelloWorldPluginSupport_destroy_dataI,
        NULL , NULL );
    if (epd == NULL) {
        return NULL;
    }
    if (endpoint_info->endpointKind == PRES_TYPEPLUGIN_ENDPOINT_WRITER) {
        serializedSampleMaxSize = HelloWorldPlugin_get_serialized_sample_max_size(
            epd,RTI_FALSE,RTI_CDR_ENCAPSULATION_ID_CDR_BE,0);
        PRESTypePluginDefaultEndpointData_setMaxSizeSerializedSample(epd, serializedSampleMaxSize);
        if (PRESTypePluginDefaultEndpointData_createWriterPool(
            epd,
            endpoint_info,
            (PRESTypePluginGetSerializedSampleMaxSizeFunction)
            HelloWorldPlugin_get_serialized_sample_max_size, epd,
            (PRESTypePluginGetSerializedSampleSizeFunction)
            PRESTypePlugin_interpretedGetSerializedSampleSize,
            epd) == RTI_FALSE) {
            PRESTypePluginDefaultEndpointData_delete(epd);
            return NULL;
        }
    }
    return epd;
```

```
}
void
HelloWorldPlugin_on_endpoint_detached(
    PRESTypePluginEndpointData endpoint_data)
{
    PRESTypePluginDefaultEndpointData_delete(endpoint_data);
}
void
HelloWorldPlugin_return_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
    void *handle)
{
    HelloWorld_finalize_optional_members(sample, RTI_TRUE);
    PRESTypePluginDefaultEndpointData_returnSample(
        endpoint_data, sample, handle);
}
void HelloWorldPlugin_finalize_optional_members(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld* sample,
    RTIBool deletePointers)
{
    RTIOsapiUtility_unusedParameter(endpoint_data);
    HelloWorld_finalize_optional_members(
        sample, deletePointers);
}
RTIBool
HelloWorldPlugin_copy_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *dst,
    const HelloWorld *src)
{
    RTIOsapiUtility_unusedParameter(endpoint_data);
    return HelloWorldPluginSupport_copy_data(dst,src);
}
/* ------------------------------------------------------------------------
(De)Serialize functions:
* ------------------------------------------------------------------------ */
unsigned int
HelloWorldPlugin_get_serialized_sample_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);
RTIBool
HelloWorldPlugin_serialize_to_cdr_buffer_ex(
    char *buffer,
    unsigned int *length,
    const HelloWorld *sample,
    DDS_DataRepresentationId_t representation)
{
    RTIEncapsulationId encapsulationId = RTI_CDR_ENCAPSULATION_ID_INVALID;
    struct RTICdrStream cdrStream;
    struct PRESTypePluginDefaultEndpointData epd;
    RTIBool result;
    struct PRESTypePluginDefaultParticipantData pd;
    struct RTIXCdrTypePluginProgramContext defaultProgramContext =
    RTIXCdrTypePluginProgramContext_INTIALIZER;
    struct PRESTypePlugin plugin;
    if (length == NULL) {
        return RTI_FALSE;
    }
    RTIOsapiMemory_zero(&epd, sizeof(struct PRESTypePluginDefaultEndpointData));
    epd.programContext = defaultProgramContext;
    epd._participantData = &pd;
    epd.typePlugin = &plugin;
    epd.programContext.endpointPluginData = &epd;
    plugin.typeCode = (struct RTICdrTypeCode *)
    HelloWorld_get_typecode();
    pd.programs = HelloWorldPlugin_get_programs();
    if (pd.programs == NULL) {
        return RTI_FALSE;
    }
    encapsulationId = DDS_TypeCode_get_native_encapsulation(
        (DDS_TypeCode *) plugin.typeCode,
        representation);
    if (encapsulationId == RTI_CDR_ENCAPSULATION_ID_INVALID) {
        return RTI_FALSE;
    }
    epd._maxSizeSerializedSample =
    HelloWorldPlugin_get_serialized_sample_max_size(
```

```
        (PRESTypePluginEndpointData)&epd,
        RTI_TRUE,
        encapsulationId,
        0);
    if (buffer == NULL) {
        *length =
        PRESTypePlugin_interpretedGetSerializedSampleSize(
            (PRESTypePluginEndpointData)&epd,
            RTI_TRUE,
            encapsulationId,
            0,
            sample);
        if (*length == 0) {
            return RTI_FALSE;
        }
        return RTI_TRUE;
    }
    RTICdrStream_init(&cdrStream);
    RTICdrStream_set(&cdrStream, (char *)buffer, *length);
    result = PRESTypePlugin_interpretedSerialize(
        (PRESTypePluginEndpointData)&epd,
        sample,
        &cdrStream,
        RTI_TRUE,
        encapsulationId,
        RTI_TRUE,
        NULL);
    *length = (unsigned int) RTICdrStream_getCurrentPositionOffset(&cdrStream);
    return result;
}
RTIBool
HelloWorldPlugin_serialize_to_cdr_buffer(
    char *buffer,
    unsigned int *length,
    const HelloWorld *sample)
{
    return HelloWorldPlugin_serialize_to_cdr_buffer_ex(
        buffer,
        length,
        sample,
        DDS_AUTO_DATA_REPRESENTATION);
}
RTIBool
HelloWorldPlugin_deserialize_from_cdr_buffer(
    HelloWorld *sample,
    const char * buffer,
    unsigned int length)
{
    struct RTICdrStream cdrStream;
    struct PRESTypePluginDefaultEndpointData epd;
    struct RTIXCdrTypePluginProgramContext defaultProgramContext =
    RTIXCdrTypePluginProgramContext_INTIALIZER;
    struct PRESTypePluginDefaultParticipantData pd;
    struct PRESTypePlugin plugin;
    epd.programContext = defaultProgramContext;
    epd._participantData = &pd;
    epd.typePlugin = &plugin;
    epd.programContext.endpointPluginData = &epd;
    plugin.typeCode = (struct RTICdrTypeCode *)
    HelloWorld_get_typecode();
    pd.programs = HelloWorldPlugin_get_programs();
    if (pd.programs == NULL) {
        return RTI_FALSE;
    }
    RTIXCdrSampleAssignabilityProperty_setFromGlobalComplianceMask(
        &epd._assignabilityProperty);
    RTICdrStream_init(&cdrStream);
    RTICdrStream_set(&cdrStream, (char *)buffer, length);
    HelloWorld_finalize_optional_members(sample, RTI_TRUE);
    return PRESTypePlugin_interpretedDeserialize(
        (PRESTypePluginEndpointData)&epd, sample,
        &cdrStream, RTI_TRUE, RTI_TRUE,
        NULL);
}
#if !defined(NDDS_STANDALONE_TYPE)
DDS_ReturnCode_t
HelloWorldPlugin_data_to_string(
    const HelloWorld *sample,
    char *_str,
    DDS_UnsignedLong *str_size,
    const struct DDS_PrintFormatProperty *property)
```

```
{
    DDS_DynamicData *data = NULL;
    char *buffer = NULL;
    unsigned int length = 0;
    struct DDS_PrintFormat printFormat;
    DDS_ReturnCode_t retCode = DDS_RETCODE_ERROR;
    if (sample == NULL) {
        return DDS_RETCODE_BAD_PARAMETER;
    }
    if (str_size == NULL) {
        return DDS_RETCODE_BAD_PARAMETER;
    }
    if (property == NULL) {
        return DDS_RETCODE_BAD_PARAMETER;
    }
    if (!HelloWorldPlugin_serialize_to_cdr_buffer(
        NULL,
        &length,
        sample)) {
        return DDS_RETCODE_ERROR;
    }
    RTIOsapiHeap_allocateBuffer(&buffer, length, RTI_OSAPI_ALIGNMENT_DEFAULT);
    if (buffer == NULL) {
        return DDS_RETCODE_ERROR;
    }
    if (!HelloWorldPlugin_serialize_to_cdr_buffer(
        buffer,
        &length,
        sample)) {
        RTIOsapiHeap_freeBuffer(buffer);
        return DDS_RETCODE_ERROR;
    }
    data = DDS_DynamicData_new(
        HelloWorld_get_typecode(),
        &DDS_DYNAMIC_DATA_PROPERTY_DEFAULT);
    if (data == NULL) {
        RTIOsapiHeap_freeBuffer(buffer);
        return DDS_RETCODE_ERROR;
    }
    retCode = DDS_DynamicData_from_cdr_buffer(data, buffer, length);
    if (retCode != DDS_RETCODE_OK) {
        RTIOsapiHeap_freeBuffer(buffer);
        DDS_DynamicData_delete(data);
        return retCode;
    }
    retCode = DDS_PrintFormatProperty_to_print_format(
        property,
        &printFormat);
    if (retCode != DDS_RETCODE_OK) {
        RTIOsapiHeap_freeBuffer(buffer);
        DDS_DynamicData_delete(data);
        return retCode;
    }
    retCode = DDS_DynamicDataFormatter_to_string_w_format(
        data,
        _str,
        str_size,
        &printFormat);
    if (retCode != DDS_RETCODE_OK) {
        RTIOsapiHeap_freeBuffer(buffer);
        DDS_DynamicData_delete(data);
        return retCode;
    }
    RTIOsapiHeap_freeBuffer(buffer);
    DDS_DynamicData_delete(data);
    return DDS_RETCODE_OK;
}
#endif
unsigned int
HelloWorldPlugin_get_serialized_sample_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment)
{
    unsigned int size;
    RTIBool overflow = RTI_FALSE;
    size = PRESTypePlugin_interpretedGetSerializedSampleMaxSize(
        endpoint_data,&overflow,include_encapsulation,encapsulation_id,current_alignment);
    if (overflow) {
        size = RTI_CDR_MAX_SERIALIZED_SIZE;
```

```
    }
    return size;
}
/* ----------------------------------------------------------------------------------
Key Management functions:
* ---------------------------------------------------------------------------------- */
PRESTypePluginKeyKind
HelloWorldPlugin_get_key_kind(void)
{
    return PRES_TYPEPLUGIN_NO_KEY;
}
RTIBool HelloWorldPlugin_deserialize_key(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld **sample,
    RTIBool * drop_sample,
    struct RTICdrStream *cdrStream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_key,
    void *endpoint_plugin_qos)
{
    RTIBool result;
    RTIOsapiUtility_unusedParameter(drop_sample);
    /*  Depending on the type and the flags used in rtiddsgen, coverity may detect
    that sample is always null. Since the case is very dependant on
    the IDL/XML and the configuration we keep the check for safety.
    */
    result= PRESTypePlugin_interpretedDeserializeKey(
        endpoint_data,
        /* coverity[check_after_deref] */
        (sample != NULL) ? *sample : NULL,
        cdrStream,
        deserialize_encapsulation,
        deserialize_key,
        endpoint_plugin_qos);
    return result;
}
unsigned int
HelloWorldPlugin_get_serialized_key_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment)
{
    unsigned int size;
    RTIBool overflow = RTI_FALSE;
    size = PRESTypePlugin_interpretedGetSerializedKeyMaxSize(
        endpoint_data,&overflow,include_encapsulation,encapsulation_id,current_alignment);
    if (overflow) {
        size = RTI_CDR_MAX_SERIALIZED_SIZE;
    }
    return size;
}
unsigned int
HelloWorldPlugin_get_serialized_key_max_size_for_keyhash(
    PRESTypePluginEndpointData endpoint_data,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment)
{
    unsigned int size;
    RTIBool overflow = RTI_FALSE;
    size = PRESTypePlugin_interpretedGetSerializedKeyMaxSizeForKeyhash(
        endpoint_data,
        &overflow,
        encapsulation_id,
        current_alignment);
    if (overflow) {
        size = RTI_CDR_MAX_SERIALIZED_SIZE;
    }
    return size;
}
struct RTIXCdrInterpreterPrograms * HelloWorldPlugin_get_programs(void)
{
    struct RTIXCdrInterpreterProgramsGenProperty programProperty =
    RTIXCdrInterpreterProgramsGenProperty_INITIALIZER;
    struct RTIXCdrInterpreterPrograms *retPrograms = NULL;
    programProperty.generateWithOnlyKeyFields = RTI_XCDR_FALSE;
    programProperty.generateV1Encapsulation = RTI_XCDR_TRUE;
    programProperty.generateV2Encapsulation = RTI_XCDR_TRUE;
    programProperty.resolveAlias = RTI_XCDR_TRUE;
    programProperty.inlineStruct = RTI_XCDR_TRUE;
    programProperty.optimizeEnum = RTI_XCDR_TRUE;
```

```
    programProperty.unboundedSize = RTIXCdrLong_MAX;
    retPrograms =
    DDS_TypeCodeFactory_assert_programs_in_global_list(
        DDS_TypeCodeFactory_get_instance(),
        HelloWorld_get_typecode(),
        &programProperty,
        RTI_XCDR_SER_PROGRAM
        | RTI_XCDR_DESER_PROGRAM
        | RTI_XCDR_GET_MAX_SER_SIZE_PROGRAM
        | RTI_XCDR_GET_SER_SIZE_PROGRAM);
    return retPrograms;
}
/* ------------------------------------------------------------------------
* Plug-in Installation Methods
* ------------------------------------------------------------------------ */
struct PRESTypePlugin *HelloWorldPlugin_new(void)
{
    struct PRESTypePlugin *plugin = NULL;
    const struct PRESTypePluginVersion PLUGIN_VERSION =
    PRES_TYPE_PLUGIN_VERSION_2_0;
    RTIOsapiHeap_allocateStructure(
        &plugin, struct PRESTypePlugin);
    if (plugin == NULL) {
        return NULL;
    }
    plugin->version = PLUGIN_VERSION;
    /* set up parent's function pointers */
    plugin->onParticipantAttached =
    (PRESTypePluginOnParticipantAttachedCallback)
    HelloWorldPlugin_on_participant_attached;
    plugin->onParticipantDetached =
    (PRESTypePluginOnParticipantDetachedCallback)
    HelloWorldPlugin_on_participant_detached;
    plugin->onEndpointAttached =
    (PRESTypePluginOnEndpointAttachedCallback)
    HelloWorldPlugin_on_endpoint_attached;
    plugin->onEndpointDetached =
    (PRESTypePluginOnEndpointDetachedCallback)
    HelloWorldPlugin_on_endpoint_detached;
    plugin->copySampleFnc =
    (PRESTypePluginCopySampleFunction)
    HelloWorldPlugin_copy_sample;
    plugin->createSampleFnc =
    (PRESTypePluginCreateSampleFunction)
    HelloWorldPlugin_create_sample;
    plugin->destroySampleFnc =
    (PRESTypePluginDestroySampleFunction)
    HelloWorldPlugin_destroy_sample;
    plugin->finalizeOptionalMembersFnc =
    (PRESTypePluginFinalizeOptionalMembersFunction)
    HelloWorldPlugin_finalize_optional_members;
    plugin->serializeFnc =
    (PRESTypePluginSerializeFunction) PRESTypePlugin_interpretedSerialize;
    plugin->deserializeFnc =
    (PRESTypePluginDeserializeFunction) PRESTypePlugin_interpretedDeserializeWithAlloc;
    plugin->getSerializedSampleMaxSizeFnc =
    (PRESTypePluginGetSerializedSampleMaxSizeFunction)
    HelloWorldPlugin_get_serialized_sample_max_size;
    plugin->getSerializedSampleMinSizeFnc =
    (PRESTypePluginGetSerializedSampleMinSizeFunction)
    PRESTypePlugin_interpretedGetSerializedSampleMinSize;
    plugin->getDeserializedSampleMaxSizeFnc = NULL;
    plugin->getSampleFnc =
    (PRESTypePluginGetSampleFunction)
    HelloWorldPlugin_get_sample;
    plugin->returnSampleFnc =
    (PRESTypePluginReturnSampleFunction)
    HelloWorldPlugin_return_sample;
    plugin->getKeyKindFnc =
    (PRESTypePluginGetKeyKindFunction)
    HelloWorldPlugin_get_key_kind;
    /* These functions are only used for keyed types. As this is not a keyed
    type they are all set to NULL
    */
    plugin->serializeKeyFnc = NULL ;
    plugin->deserializeKeyFnc = NULL;
    plugin->getKeyFnc = NULL;
    plugin->returnKeyFnc = NULL;
    plugin->instanceToKeyFnc = NULL;
    plugin->keyToInstanceFnc = NULL;
    plugin->getSerializedKeyMaxSizeFnc = NULL;
```

```
    plugin->instanceToKeyHashFnc = NULL;
    plugin->serializedSampleToKeyHashFnc = NULL;
    plugin->serializedKeyToKeyHashFnc = NULL;
    #ifdef NDDS_STANDALONE_TYPE
    plugin->typeCode = NULL;
    #else
    plugin->typeCode =  (struct RTICdrTypeCode *)HelloWorld_get_typecode();
    #endif
    plugin->languageKind = PRES_TYPEPLUGIN_DDS_TYPE;
    /* Serialized buffer */
    plugin->getBuffer =
    (PRESTypePluginGetBufferFunction)
    HelloWorldPlugin_get_buffer;
    plugin->returnBuffer =
    (PRESTypePluginReturnBufferFunction)
    HelloWorldPlugin_return_buffer;
    plugin->getBufferWithParams = NULL;
    plugin->returnBufferWithParams = NULL;
    plugin->getSerializedSampleSizeFnc =
    (PRESTypePluginGetSerializedSampleSizeFunction)
    PRESTypePlugin_interpretedGetSerializedSampleSize;
    plugin->getWriterLoanedSampleFnc = NULL;
    plugin->returnWriterLoanedSampleFnc = NULL;
    plugin->returnWriterLoanedSampleFromCookieFnc = NULL;
    plugin->validateWriterLoanedSampleFnc = NULL;
    plugin->setWriterLoanedSampleSerializedStateFnc = NULL;
    plugin->endpointTypeName = HelloWorldTYPENAME;
    plugin->isMetpType = RTI_FALSE;
    plugin->isRecursiveType = RTI_FALSE;
    return plugin;
}
void
HelloWorldPlugin_delete(struct PRESTypePlugin *plugin)
{
    RTIOsapiHeap_freeStructure(plugin);
}
#undef RTI_CDR_CURRENT_SUBMODULE
```

5.16 HelloWorldSupport.c

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorldSupport.h]

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.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 HelloWorldSupport_1436885484_h
#define HelloWorldSupport_1436885484_h
/* Uses */
#include "HelloWorld.h"
#ifndef ndds_c_h
#include "ndds/ndds_c.h"
#endif
#ifdef __cplusplus
extern "C" {
    #endif
    #if (defined(RTI_WIN32) || defined (RTI_WINCE) || defined(RTI_INTIME)) && defined(NDDS_USER_DLL_EXPORT)
    #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
    DDS_TYPESUPPORT_C(HelloWorldTypeSupport, HelloWorld);
    DDS_DATAWRITER_WITH_DATA_CONSTRUCTOR_METHODS_C(HelloWorldDataWriter, HelloWorld);
    DDS_DATAREADER_C(HelloWorldDataReader, HelloWorldSeq, HelloWorld);
    #if defined(NDDS_USER_DLL_EXPORT) || defined(NDDS_USER_SYMBOL_EXPORT)
```

```
    #undef NDDSUSERDllExport
    #define NDDSUSERDllExport
    #endif
    #ifdef __cplusplus
}
#endif
#endif  /* HelloWorldSupport_1436885484_h */
```

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorldSupport.c]

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.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.
*/
#include "HelloWorldSupport.h"
#include "HelloWorldPlugin.h"
/* ========================================================================= */
/* ----------------------------------------------------------------- */
/* DDSDataWriter
*/
/* Requires */
#define TTYPENAME   HelloWorldTYPENAME
/* Defines */
#define TDataWriter HelloWorldDataWriter
#define TData       HelloWorld
#define RTI_ENABLE_TDATAWRITER_DATA_CONSTRUCTOR_METHODS
#include "dds_c/generic/dds_c_data_TDataWriter.gen"
#undef RTI_ENABLE_TDATAWRITER_DATA_CONSTRUCTOR_METHODS
#undef TDataWriter
#undef TData
#undef TTYPENAME
/* ----------------------------------------------------------------- */
/* DDSDataReader
*/
/* Requires */
#define TTYPENAME   HelloWorldTYPENAME
/* Defines */
#define TDataReader HelloWorldDataReader
#define TDataSeq    HelloWorldSeq
#define TData       HelloWorld
#define RTI_ENABLE_TDATAREADER_DATA_CONSISTENCY_CHECK_METHOD
#include "dds_c/generic/dds_c_data_TDataReader.gen"
#undef RTI_ENABLE_TDATAREADER_DATA_CONSISTENCY_CHECK_METHOD
#undef TDataReader
#undef TDataSeq
#undef TData
#undef TTYPENAME
/* ----------------------------------------------------------------- */
/* TypeSupport

«IMPLEMENTATION »

Requires:  TTYPENAME,
TPlugin_new
TPlugin_delete
Defines:   TTypeSupport, TData, TDataReader, TDataWriter
*/
/* Requires */
#define TTYPENAME    HelloWorldTYPENAME
#define TPlugin_new  HelloWorldPlugin_new
#define TPlugin_delete  HelloWorldPlugin_delete
/* Defines */
#define TTypeSupport HelloWorldTypeSupport
#define TData        HelloWorld
#define TDataReader  HelloWorldDataReader
#define TDataWriter  HelloWorldDataWriter
#define TGENERATE_SER_CODE
#ifndef NDDS_STANDALONE_TYPE
#define TGENERATE_TYPECODE
#endif
#include "dds_c/generic/dds_c_data_TTypeSupport.gen"
#undef TTypeSupport
#undef TData
#undef TDataReader
#undef TDataWriter
#ifndef NDDS_STANDALONE_TYPE
```

```
#undef TGENERATE_TYPECODE
#endif
#undef TGENERATE_SER_CODE
#undef TTYPENAME
#undef TPlugin_new
#undef TPlugin_delete
```

Chapter 6

Data Structure Documentation

6.1 NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t Struct Reference

Connection liveliness settings.

Data Fields

- RTI_INT32 **enable**
- RTI_INT32 **lease_duration**
- RTI_INT32 **assertions_per_lease_duration**

6.1.1 Detailed Description

Connection liveliness settings.

6.1.2 Field Documentation

6.1.2.1 enable

```
RTI_INT32 NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t::enable
```

Enables the liveliness mechanism for the control connection.

[default] 0 (disabled)

6.1.2.2 lease_duration

```
RTI_INT32 NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t::lease_duration
```

In seconds, the timeout by which the connection liveliness must be asserted or the connection will be considered not alive. Please note that this is used also as the period between connection liveliness checks. Therefore, the maximum time before a connection is marked as not alive is 2∗connection_liveliness_lease_duration.

[default] 10 seconds

6.1.2.3 assertions_per_lease_duration

```
RTI_INT32 NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t::assertions_per_lease_duration
```

The number of connection assertions/heartbeats sent per lease duration.

[default] 3

6.2 NDDS_Transport_TCPv4_EventThreadSettings_t Struct Reference

Event thread settings.

Data Fields

- RTI_INT32 **priority**
- RTI_INT32 **stack_size**
- RTI_INT32 **options**

6.2.1 Detailed Description

Event thread settings.

6.2.2 Field Documentation

6.2.2.1 priority

```
RTI_INT32 NDDS_Transport_TCPv4_EventThreadSettings_t::priority
```

The priority associated to the TCPv4 Plugin event thread.

[default] THREAD_PRIORITY_DEFAULT

6.2.2.2 stack_size

```
RTI_INT32 NDDS_Transport_TCPv4_EventThreadSettings_t::stack_size
```

The stack size associated to the TCPv4 Plugin event thread.

[default] THREAD_STACK_SIZE_DEFAULT

6.2.2.3 options

```
RTI_INT32 NDDS_Transport_TCPv4_EventThreadSettings_t::options
```

The thread-options associated to the TCPv4 Plugin event thread.

[default] PRIORITY_ENFORCE | STDIO

6.3 NDDS_Transport_TCPv4_Property_t Struct Reference

Configurable TCP Transport-Plugin properties.

Data Fields

- 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_nonrunning_interfaces**

  Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.
- RTI_INT32 **transport_priority_mask**

  Mask for the transport priority field.
- RTI_INT32 **transport_priority_mapping_low**

  Sets the low value of the output range to IPv4 TOS.
- RTI_INT32 **transport_priority_mapping_high**

  Sets the high value of the output range to IPv4 TOS.
- RTI_INT32 **force_interface_poll_detection**

  Sets the interface tracker detection kind to polling for detecting changes in the interfaces. Only used when using Transport TCP Plugin LAN mode (class-id NDDS_TRANSPORT_CLASSID_TCPV4_LAN).
- RTI_UINT32 **interface_poll_period**

  Sets the interface polling period for detecting changes in the interfaces. Only used when using Transport TCP Plugin LAN mode (class-id NDDS_TRANSPORT_CLASSID_TCPV4_LAN).
- RTI_INT32 **server_socket_backlog**

Determines what is the maximum length of the queue of pending connections.

- **NDDS_Transport_Address_t public_address**

  Public locator (IP address and port) of the transport instantiation.

- char * **bind_interface_address**

  Interface IP address for the transport sockets.

- RTI_INT32 **server_bind_port**

  Private IP port (inside the LAN) used by the transport to accept TCP connections.

- struct **TransportAllocationSettings_t read_buffer_allocation**

  Allocation settings applied to read buffers.

- struct **TransportAllocationSettings_t write_buffer_allocation**

  *Allocation settings applied to buffers used for asynchronous (nonblocking) send. To enable asynchronous send, set the field **force_asynchronous_send** (p. 83) to 1.*

- RTI_INT32 **shared_write_buffer_allocation**

  Determines whether the pool of buffers created with asynchronous (non-blocking) send is shared or exclusive per TCP connection.

- struct **TransportAllocationSettings_t control_buffer_allocation**

  Allocation settings applied to buffers used to serialize and send control messages.

- struct **TransportAllocationSettings_t control_message_allocation**

  Allocation settings applied to control messages.

- struct **TransportAllocationSettings_t control_attribute_allocation**

  Allocation settings applied to control messages attributes.

- RTI_INT32 **force_asynchronous_send**

  Controls whether the plugin will send data synchronously or asynchronously.

- **NDDS_Transport_TCPv4_OnConnectionEstablishedCallback on_connection_established**
- **NDDS_Transport_TCPv4_OnConnectionLostCallback on_connection_lost**
- RTI_INT32 **max_packet_size**

  The maximum size of a TCP segment.

- RTI_INT32 **enable_keep_alive**

  Configures the sending of KEEP_ALIVE messages in TCP.

- RTI_INT32 **keep_alive_time**

  Specifies the interval of inactivity in seconds that causes TCP to generate a KEEP_ALIVE message.

- RTI_INT32 **keep_alive_interval**

  Specifies the interval in seconds between KEEP_ALIVE retries.

- RTI_INT32 **keep_alive_retry_count**

  The maximum number of KEEP_ALIVE retries before dropping the connection.

- RTI_INT32 **user_timeout**
- struct **NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t connection_liveliness_settings**
- struct **NDDS_Transport_TCPv4_EventThreadSettings_t event_thread_settings**
- RTI_INT32 **disable_nagle**

  Disables the TCP nagle algorithm.

- RTI_INT32 **logging_verbosity_bitmap**

  Bitmap that specifies the verbosity of log messages from the transport.

- RTI_INT32 **security_logging_verbosity_bitmap**

  Bitmap that specifies the verbosity of log messages generated by OpenSSL.

- **NDDS_Transport_TCPv4_SocketMonitoringKind_t socket_monitoring_kind**

  Configures the socket monitoring API used by the transport.

- struct **NDDS_Transport_TCPv4_WindowsIOCPSettings_t windows_iocp_settings**

*Configures I/O completion ports when **socket_monitoring_kind** (p. 87) is set to **NDDS_TRANSPORT_TCPV4_↩ SOCKET_MONITORING_KIND_WINDOWS_IOCP** (p. 19).*

- RTI_INT32 **send_crc**

  Tells the plugin to calculate and send the CRC32 for the TCP message payload.
- RTI_INT32 **force_crc_check**

  Tells the plugin to force the checking of the checksum. By default, the checksum will be checked only if present.
- RTI_INT32 **outstanding_connection_cookies**

  Maximum number of outstanding connection cookies allowed by the transport when acting as server.
- RTI_INT32 **outstanding_connection_cookies_life_span**

  Maximum lifespan (in seconds) of the cookies associated with pending connections.
- RTI_INT32 **send_max_wait_sec**

  Maximum number of seconds a low-level TCP sendto() function is allowed to block.
- RTI_INT32 **client_connection_negotiation_timeout**

  Timeout (in seconds) for negotiating a client data connection.
- RTI_INT32 **server_connection_negotiation_timeout**

  Timeout (in seconds) for negotiating a server data connection.
- RTI_INT32 **initial_handshake_timeout**

  Timeout (in seconds) for the initial handshake for a connection.
- struct **RTITLS_OpenSSL_Configuration tls**

  OpenSSL TLS parameters.
- RTI_INT32 **disable_interface_tracking**

  If this variable is set, the automatic change detection over the system network interfaces will be disabled.

6.3.1 Detailed Description

Configurable TCP Transport-Plugin properties.

For communications within a LAN you should initialize the object as follows:

```
prop.parent.classid = NDDS_TRANSPORT_CLASSID_TCPV4_LAN;
```

If you want to use the transport in WAN configuration, you must also define the public_address and sever_bind_port:

```
prop.parent.classid = NDDS_TRANSPORT_CLASSID_TCPV4_WAN;
NDDS_Transport_TCPv4_Plugin_stringToTransportAddress(
        &prop.public_address,
        "142.123.123.111:7890",
        0);
prop.server_bind_port = 7400;
```

Remember that the public_address is the public address that your network gateway is exposing to a remote peer. It is not the local network address where the application is running.

6.3.2 Field Documentation

6.3.2.1 parent

struct **NDDS_Transport_Property_t** NDDS_Transport_TCPv4_Property_t::parent

Generic properties of all transport plugins.

[default] Refer to the property default NDDS_Transport_TCPv4_Plugin::NDDS_TRANSPORT_TCPV4_PROPERTY↩ _DEFAULT_LAN

6.3.2.2 send_socket_buffer_size

RTI_INT32 NDDS_Transport_TCPv4_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** or **NDDS_↩ TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT** (p. 12).

The maximum value is operating system-dependent.

[default] NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT (p. 12)

6.3.2.3 recv_socket_buffer_size

RTI_INT32 NDDS_Transport_TCPv4_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 parent.message_size_max or **NDDS_TRANSPORT_TCPV4_SOCKET_↩ BUFFER_SIZE_OS_DEFAULT** (p. 12).

The maximum value is operating-system dependent.

[default] NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT (p. 12)

6.3.2.4 ignore_loopback_interface

RTI_INT32 NDDS_Transport_TCPv4_Property_t::ignore_loopback_interface

Prevents the transport plugin from using the IP loopback interface.

This property is ignored when parent.classid is equal to **NDDS_TRANSPORT_CLASSID_TCPV4_WAN**.

Two values are allowed:

- 0: Enable local traffic via this plugin. The plugin will only use and report the IP loopback interface only if there are no other network interfaces (NICs) up on the system.
- 1: Disable local traffic via this plugin. This means "do not use the IP loopback interface, even if no NICs are discovered." This setting is useful when you want applications running on the same node to use a more efficient plugin like shared memory instead of the IP loopback.

[default] 1

6.3.2.5 ignore_nonrunning_interfaces

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::ignore_nonrunning_interfaces
```

Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.

This property has no effect. Non-RUNNING Interfaces are ignored until they change their status to RUNNING, unless **disable_interface_tracking** (p. 90) 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 which 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 no link is 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

6.3.2.6 transport_priority_mask

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::transport_priority_mask
```

Mask for the transport priority field.

This is used in conjunction with **transport_priority_mapping_low** (p. 79) and **transport_priority_mapping_high** (p. 80) to define the mapping from DDS transport priority (see **TRANSPORT_PRIORITY**) 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

6.3.2.7 transport_priority_mapping_low

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::transport_priority_mapping_low
```

Sets the low value of the output range to IPv4 TOS.

This is used in conjunction with **transport_priority_mask** (p. 79) and **transport_priority_mapping_high** (p. 80) to define the mapping from 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.

See also

transport_priority_mask (p. 79)

[default] 0

6.3.2.8 transport_priority_mapping_high

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::transport_priority_mapping_high
```

Sets the high value of the output range to IPv4 TOS.

This is used in conjunction with **transport_priority_mask** (p. 79) and **transport_priority_mapping_low** (p. 79) to define the mapping from 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.

See also

transport_priority_mask (p. 79)

[default] 0xFF

6.3.2.9 force_interface_poll_detection

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::force_interface_poll_detection
```

Sets the interface tracker detection kind to polling for detecting changes in the interfaces. Only used when using Transport TCP Plugin LAN mode (class-id NDDS_TRANSPORT_CLASSID_TCPV4_LAN).

See **NDDS_Transport_UDPv4_Property_t::force_interface_poll_detection** for more details.

6.3.2.10 interface_poll_period

```
RTI_UINT32 NDDS_Transport_TCPv4_Property_t::interface_poll_period
```

Sets the interface polling period for detecting changes in the interfaces. Only used when using Transport TCP Plugin LAN mode (class-id NDDS_TRANSPORT_CLASSID_TCPV4_LAN).

See **NDDS_Transport_UDPv4_Property_t::interface_poll_period** for more details.

6.3.2.11 server_socket_backlog

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::server_socket_backlog
```

Determines what is the maximum length of the queue of pending connections.

[default] NDDS_TRANSPORT_TCPV4_SERVER_SOCKET_BACKLOG_DEFAULT (p. 13)

6.3.2.12 public_address

```
NDDS_Transport_Address_t NDDS_Transport_TCPv4_Property_t::public_address
```

Public locator (IP address and port) of the transport instantiation.

Use the function **NDDS_Transport_TCPv4_Plugin_stringToTransportAddress** (p. 20) to compose the transport locator from a string form.

This field is used and required only when **NDDS_Transport_Property_t::classid** is set to **NDDS_TRANSPORT_↩ CLASSID_TCPV4_WAN**.

The public IP address and port are necessary to support communication over WAN that involves Network Address Translators (NATs).

Typically, the address is the public address of the IP router that provides access to the WAN. The port is the IP router port that is used to reach the private **server_bind_port** (p. 81) inside the LAN from the outside.

See also

NDDS_Transport_TCPv4_Property_t::server_bind_port (p. 81)

[default] NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER

6.3.2.13 bind_interface_address

```
char* NDDS_Transport_TCPv4_Property_t::bind_interface_address
```

Interface IP address for the transport sockets.

The TCP transport can be configured to bind all sockets to a specified interface.

If the value is NULL, the sockets will be bound to the special IP address INADDR_ANY. This address allows the sockets to receive packets destined to any of the interfaces.

This field should be set in multi-homed systems communicating across NAT routers.

[default] NULL

6.3.2.14 server_bind_port

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::server_bind_port
```

Private IP port (inside the LAN) used by the transport to accept TCP connections.

If this property is set to zero, the transport will disable the internal server socket, making it impossible for external peers to connect to this node. In this case, the node is considered unreachable and will only communicate using the asymmetric mode with other (reachable) peers.

For WAN communication, this port must be forwarded to a public port in the NAT-enabled router that connects to the outer network.

The server_bind_port cannot be shared among multiple participants on a common host. On most operating systems, attempting to reuse the same server_bind_port for multiple participants on a common host will result in a "port already in use" error. However, Windows systems will not recognize if the server_bind_port is already in use, and thus care must be taken to properly configure Windows systems.

```
\see NDDS_Transport_TCPv4_Property_t#public_address

<B>[default]</B> \ref NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER
```

6.3.2.15 read_buffer_allocation

```
struct TransportAllocationSettings_t NDDS_Transport_TCPv4_Property_t::read_buffer_allocation
```

Allocation settings applied to read buffers.

These settings configure the initial number of buffers, the maximum number of buffers and the buffers to be allocated when more buffers are needed.

[default] NDDS_TRANSPORT_TCPV4_READ_BUFFER_POOL_GROWTH_POLICY_DEFAULT (p. 13)

6.3.2.16 write_buffer_allocation

```
struct TransportAllocationSettings_t NDDS_Transport_TCPv4_Property_t::write_buffer_allocation
```

Allocation settings applied to buffers used for asynchronous (nonblocking) send. To enable asynchronous send, set the field **force_asynchronous_send** (p. 83) to 1.

These settings configure the initial number of buffers, the maximum number of buffers and the buffers to be allocated when more buffers are needed.

The pool of buffers can be configured to be shared across all the TCP connections created by the TCP Transport or to be exclusive for a connection by setting the property **shared_write_buffer_allocation** (p. 82) to 1. The default value is 0.

Note that for the write buffer pool, the max_count is not set to unlimited. This is to avoid having a fast writer quickly exhaust all the available system memory, in case of a temporary network slowdown. When this write buffer pool reaches the maximum, a new message will replace the oldest message that is not currently in the process of being sent. This guarantees that new messages are prioritized, while at the same time not running into a situation in which messages are not received. Messages that are replaced and not sent may be resent later depending on the application's QoS (if the transport is used for reliable communication, the data will still be sent eventually).

[default] NDDS_TRANSPORT_TCPV4_WRITE_BUFFER_POOL_GROWTH_POLICY_DEFAULT (p. 13)

6.3.2.17 shared_write_buffer_allocation

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::shared_write_buffer_allocation
```

Determines whether the pool of buffers created with asynchronous (non-blocking) send is shared or exclusive per TCP connection.

Sharing this buffer across connections may lead to less memory consumption. However, high-throughput connections may starve low-throughput connections. This is why the default value is 0.

The size of the buffer pool can be configured using the field **write_buffer_allocation** (p. 82).

[default] 0 (allocation settings per TCP connection)

6.3.2.18 control_buffer_allocation

```
struct TransportAllocationSettings_t NDDS_Transport_TCPv4_Property_t::control_buffer_allocation
```

Allocation settings applied to buffers used to serialize and send control messages.

These settings configure the initial number of buffers, the maximum number of buffers and the buffers to be allocated when more buffers are needed.

[default] NDDS_TRANSPORT_TCPV4_CONTROL_BUFFER_POOL_GROWTH_POLICY_DEFAULT (p. 14)

6.3.2.19 control_message_allocation

```
struct TransportAllocationSettings_t NDDS_Transport_TCPv4_Property_t::control_message_allocation
```

Allocation settings applied to control messages.

These settings configure the initial number of buffers, the maximum number of buffers and the buffers to be allocated when more buffers are needed.

[default] NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_FACTORY_GROWTH_POLICY_DEFAULT (p. 14)

6.3.2.20 control_attribute_allocation

```
struct TransportAllocationSettings_t NDDS_Transport_TCPv4_Property_t::control_attribute_allocation
```

Allocation settings applied to control messages attributes.

These settings configure the initial number of attributes, the maximum number of attributes and the attributes to be allocated when more attributes are needed.

[default] NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_ATTRIBUTE_FACTORY_GROWTH_POLICY_↩ DEFAULT (p. 14)

6.3.2.21 force_asynchronous_send

`RTI_INT32 NDDS_Transport_TCPv4_Property_t::force_asynchronous_send`

Controls whether the plugin will send data synchronously or asynchronously.

When this parameter is set to 0, the TCP Transport will attempt to send data as soon as the internal send() function is called. When it is set to 1, the transport will make a copy of the data to send in an internal send buffer and enqueue it. Data will be sent as soon as the low-level socket buffer has space.

Setting this option to 0 (default) should provide better latency. However, in high-throughput scenarios, a 0 setting may cause the low-level send() function to block until the data is physically delivered to the lower socket buffer. For an application writing data at a very fast rate, the 0 setting may cause the caller thread to block if the send socket buffer is full. This could produce lower throughput in those conditions (the caller thread could prepare the next packet while waiting for the send socket buffer to become available).

The size of the buffer pool created by setting this option 1 can be configured using the field **write_buffer_allocation** (p. 82). In addition, the TCP Transport can be used to create one buffer pool per connection or a single buffer pool shared across all TCP connections by using the field **shared_write_buffer_allocation** (p. 82).

[default] 0 (synchronous send)

6.3.2.22 on_connection_established

NDDS_Transport_TCPv4_OnConnectionEstablishedCallback `NDDS_Transport_TCPv4_Property_t::on_connection_established`

Pointer to a function that is called whenever the plugin establish a connection with a remote peer.

For plugins configured to behave as servers, this function is called every time a remote client successfully establish a data communication.

For plugins configured to behave as clients, this function is called for every successful connection to a remote server.

It can be set to NULL (no notification callbacks are performed)

IMPORTANT: This function is called from the receive thread (both client/server side). if you don't return the control fast enough, the control protocol may be affected causing disconnections and/or delays in establishing connections with remote peers.

[default] NULL

6.3.2.23 on_connection_lost

NDDS_Transport_TCPv4_OnConnectionLostCallback `NDDS_Transport_TCPv4_Property_t::on_connection_lost`

Pointer to a function that is called when a previously established connection with a remote peer gets closed.

It can be set to NULL (no notification callbacks are performed)

IMPORTANT: This function is called from the receive thread (both client/server side). if you don't return the control fast enough, the control protocol may be affected causing disconnections and/or delays in establishing connections with remote peers.

[default] NULL

6.3.2.24 max_packet_size

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::max_packet_size
```

The maximum size of a TCP segment.

This parameter is only supported on Linux architectures.

By default, the maximum size of a TCP segment is based on the network MTU for destinations on a local network, or on a default 576 for destinations on non-local networks. This behavior can be changed by setting this parameter to a value between 1 and 65535.

[default] -1 (use OS default)

6.3.2.25 enable_keep_alive

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::enable_keep_alive
```

Configures the sending of KEEP_ALIVE messages in TCP.

Setting this value to 1, causes a KEEP_ALIVE packet to be sent to the remote peer if a long time passes with no other data sent or received.

This feature is implemented only on architectures that provide a lowlevel implementation of the TCP keep-alive feature.

On Windows systems, the TCP keep-alive feature can be globally enabled through the system's registry: **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Tcpip\Parameters**.

Refer to MSDN documentation for more details.

[default] 0 (disabled)

6.3.2.26 keep_alive_time

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::keep_alive_time
```

Specifies the interval of inactivity in seconds that causes TCP to generate a KEEP_ALIVE message.

This parameter is only supported on Linux architectures.

[default] -1 (OS default value)

6.3.2.27 keep_alive_interval

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::keep_alive_interval
```

Specifies the interval in seconds between KEEP_ALIVE retries.

This parameter is only supported on Linux architectures.

[default] -1 (OS default value)

6.3.2.28 keep_alive_retry_count

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::keep_alive_retry_count
```

The maximum number of KEEP_ALIVE retries before dropping the connection.

This parameter is only supported on Linux architectures.

[default] -1 (use system's default).

6.3.2.29 user_timeout

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::user_timeout
```

Changes the default OS TCP User Timeout configuration. If set to a value greater than 0, it represents the TCP User Timeout in seconds.

Currently this feature is supported only on Linux 2.6.37 and above.

[default] 0 (use system's default).

6.3.2.30 connection_liveliness_settings

```
struct NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t NDDS_Transport_TCPv4_Property_t↩
::connection_liveliness_settings
```

Configuration for the liveliness mechanism for the control connection.

[default] NDDS_TRANSPORT_TCPV4_CONNECTION_LIVELINESS_SETTINGS_DEFAULT (p. 14)

6.3.2.31 event_thread_settings

```
struct NDDS_Transport_TCPv4_EventThreadSettings_t NDDS_Transport_TCPv4_Property_t::event_thread↩
_settings
```

Configuration for the TCPv4 Event Thread.

[default] NDDS_TRANSPORT_TCPV4_EVENT_THREAD_SETTINGS_DEFAULT (p. 14)

6.3.2.32 disable_nagle

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::disable_nagle
```

Disables the TCP nagle algorithm.

When this property is set to 1, TCP segments are always sent as soon as possible, which may result in poor network utilization.

[default] 0 (disabled)

6.3.2.33 logging_verbosity_bitmap

`RTI_INT32 NDDS_Transport_TCPv4_Property_t::logging_verbosity_bitmap`

Bitmap that specifies the verbosity of log messages from the transport.

Logging values:

- -1 (0xffffffff): do not change the current verbosity
- 0x00: silence
- 0x01: fatal error
- 0x02: errors
- 0x04: warnings
- 0x08: local
- 0x10: remote
- 0x20: period
- 0x100: other (used for control protocol tracing)

You can combine these values by logically ORing them together. For example, 0x9F means all (errors, warnings, local, remote, period, and other).

Note: The logging verbosity is a global property shared across multiple instances of the TCP transport. If you create a new TCP Transport instance with logging_verbosity_bitmap different than -1, the change will affect all the other instances as well.

The default TCP transport verbosity is errors and warnings.

Note: The option of 0x80 (other) is used only for tracing the internal control protocol. Since the output is very verbose, this feature is enabled only in the debug version of the TCP Transport library.

[default] -1 (do not change default verbosity).

6.3.2.34 security_logging_verbosity_bitmap

`RTI_INT32 NDDS_Transport_TCPv4_Property_t::security_logging_verbosity_bitmap`

Bitmap that specifies the verbosity of log messages generated by OpenSSL.

Logging values:

- -1 (0xffffffff): Inherit logging_verbosity_bitmap value
- 0x00: silence
- 0x01: fatal error
- 0x02: errors
- 0x04: warnings
- 0x08: local
- 0x10: remote
- 0x20: period

You can combine these values by logically ORing them together.

[default] -1 (Inherit logging_verbosity_bitmap value)

6.3.2.35 socket_monitoring_kind

NDDS_Transport_TCPv4_SocketMonitoringKind_t NDDS_Transport_TCPv4_Property_t::socket_monitoring_↩kind

Configures the socket monitoring API used by the transport.

[default] NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_SELECT (p. 19) **[default] NDDS_↩TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_WINDOWS_IOCP** (p. 19) on Windows **[default] NDDS_↩TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_POLL** (p. 19) on Linux

6.3.2.36 windows_iocp_settings

struct **NDDS_Transport_TCPv4_WindowsIOCPSettings_t** NDDS_Transport_TCPv4_Property_t::windows_iocp↩_settings

Configures I/O completion ports when **socket_monitoring_kind** (p. 87) is set to **NDDS_TRANSPORT_TCPV4_↩SOCKET_MONITORING_KIND_WINDOWS_IOCP** (p. 19).

[default] NDDS_TRANSPORT_TCPV4_WINDOWS_IOCP_SETTINGS_DEFAULT (p. 15)

6.3.2.37 send_crc

RTI_INT32 NDDS_Transport_TCPv4_Property_t::send_crc

Tells the plugin to calculate and send the CRC32 for the TCP message payload.

[default] 0

6.3.2.38 force_crc_check

RTI_INT32 NDDS_Transport_TCPv4_Property_t::force_crc_check

Tells the plugin to force the checking of the checksum. By default, the checksum will be checked only if present.

[default] 0

6.3.2.39 outstanding_connection_cookies

RTI_INT32 NDDS_Transport_TCPv4_Property_t::outstanding_connection_cookies

Maximum number of outstanding connection cookies allowed by the transport when acting as server.

A connection cookie is a token provided by a server to a client; it is used to establish a data connection. Until the data connection is established, the cookie cannot be reused by the server.

To avoid wasting memory, it is good practice to set a cap to the maximum number of connection cookies (pending connections).

When the maximum value is reached, a client will not be able to connect to the server until new cookies become available.

[default] 100

6.3.2.40 outstanding_connection_cookies_life_span

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::outstanding_connection_cookies_life_span
```

Maximum lifespan (in seconds) of the cookies associated with pending connections.

If a client does not connect to the server before the lifespan of its cookie expires, it will have to request a new cookie.

Range: 1 second or higher, or -1.

[default] -1, which means an unlimited amount of time (effectively disabling the feature).

6.3.2.41 send_max_wait_sec

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::send_max_wait_sec
```

Maximum number of seconds a low-level TCP sendto() function is allowed to block.

This property controls the maximum time (in seconds) the low level sendto() is allowed to block the caller thread when the TCP send buffer becomes full.

If the bandwidth used by the transport is limited, and the sender thread try to push data faster than the OS can handle, the low level sendto() function will block the caller until there is some room available on the queue.

By limiting this delay we eliminate possibility of deadlock and increase the response time of the internal DDS thread.

This property affects both CONTROL and DATA stream, and affect only SYNCHRONOUS send operations. Asynchronous send never blocks a send operation.

For synchronous send() this property limit the time the DDS sender thread can block for a send buffer full. If is too large, DDS not only won't be able to send more data, but it won't be able to receive any more data because of an internal resource mutex.

Setting this property to 0 cause the low level function to report an immediate failure if the TCP send buffer is full.

Setting this property to -1 cause the low level function to block forever until space becomes available in the TCP buffer

[default] 3 seconds.

6.3.2.42 client_connection_negotiation_timeout

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::client_connection_negotiation_timeout
```

Timeout (in seconds) for negotiating a client data connection.

The TCP Transport Plugin requires some negotiation before establishing a connection. This property controls the maximum time (in seconds) a client data connection negotiation can remain in progress.

In particular, it controls a maximum timeout for requesting and replying to a server logical port request.

If the negotiation of a connection has not completed after the specified timeout, the negotiation will restart, and if there is an associated data connection, it will be closed. This way, the TCP Transport Plugin can retry the process of establishing and negotiating that connection.

Range: [1, 2147483647] seconds.

[default] 10 seconds

6.3.2.43 server_connection_negotiation_timeout

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::server_connection_negotiation_timeout
```

Timeout (in seconds) for negotiating a server data connection.

The TCP Transport Plugin requires some negotiation before establishing a connection. This property controls the maximum time (in seconds) a server data connection negotiation can remain in progress.

In particular, it controls a maximum timeout for requesting and replying to a client logical port request and binding the connection.

If the negotiation of a connection has not completed after the specified timeout, the negotiation will restart, and if there is an associated data connection, it will be closed. This way, the TCP Transport Plugin can retry the process of establishing and negotiating that connection.

Range: [1, 2147483647] seconds.

[default] 10 seconds

6.3.2.44 initial_handshake_timeout

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::initial_handshake_timeout
```

Timeout (in seconds) for the initial handshake for a connection.

Once a connection is established, TCP transport will exchange some information to identify itself and the connection. This process is known as the initial handshake of a connection, and if using TLS it will also exchange additional information to secure the connection.

This property controls the maximum time (in seconds) the initial handshake for a connection can remain in progress. If the handshake has not completed after the specified timeout, the connection will be closed. This way, the TCP Transport Plugin can restart the process of establishing and handshaking that connection.

Range: [1, 2147483647] seconds.

[default] 10 seconds

6.3.2.45 tls

```
struct RTITLS_OpenSSL_Configuration NDDS_Transport_TCPv4_Property_t::tls
```

OpenSSL TLS parameters.

[default] RTITLS_OpenSSL::RTITLS_OPENSSL_CONFIGURATION_DEFAULT

6.3.2.46 disable_interface_tracking

```
RTI_INT32 NDDS_Transport_TCPv4_Property_t::disable_interface_tracking
```

If this variable is set, the automatic change detection over the system network interfaces will be disabled.

See **NDDS_Transport_UDPv4_Property_t::disable_interface_tracking** for more details.

[default] 0

6.4 NDDS_Transport_TCPv4_WindowsIOCPSettings_t Struct Reference

Windows I/O completion ports settings.

Data Fields

- RTI_UINT32 **concurrency_value**

  Defines the concurrency value (number of concurrent running threads) used by I/O Completion Ports.
- RTI_UINT32 **thread_pool_size**

  Defines the thread pool size used by I/O Completion Ports.

6.4.1 Detailed Description

Windows I/O completion ports settings.

6.4.2 Field Documentation

6.4.2.1 concurrency_value

```
RTI_UINT32 NDDS_Transport_TCPv4_WindowsIOCPSettings_t::concurrency_value
```

Defines the concurrency value (number of concurrent running threads) used by I/O Completion Ports.

[default] 1

6.4.2.2 thread_pool_size

```
RTI_UINT32 NDDS_Transport_TCPv4_WindowsIOCPSettings_t::thread_pool_size
```

Defines the thread pool size used by I/O Completion Ports.

[default] 2

6.5 RTITLS_Ciphers Struct Reference

Set of TLS properties for cipher configuration.

`#include <rtitls_openssl.h>`

Data Fields

- char * **cipher_list**

  List of available TLS ciphers for TLS 1.2 or below.
- char * **ciphersuites**

  List of available TLS ciphersuites for TLS 1.3 or below.
- **DDS_Long dh_param_files_length**

  Number of DH key files supplied.
- struct **RTITLS_DHParamFile** * **dh_param_files**

  List of available DH key files.
- char * **engine_id**

  ID of OpenSSL cipher engine to request.

6.5.1 Detailed Description

Set of TLS properties for cipher configuration.

6.5.2 Field Documentation

6.5.2.1 cipher_list

`char* RTITLS_Ciphers::cipher_list`

List of available TLS ciphers for TLS 1.2 or below.

See the OpenSSL manual page for SSL_set_cipher_list(3) or ciphers(1) for more information on the format of this string. Some typical values are defined: RTITLS_Plugin::RTITLS_CIPHER_LIST_DEFAULT, RTITLS_Plugin::RTITLS↩ _CIPHER_LIST_ENCRYPT_HIGH, and RTITLS_Plugin::RTITLS_CIPHER_LIST_UNENCRYPTED.

[default] NULL

6.5.2.2 ciphersuites

```
char* RTITLS_Ciphers::ciphersuites
```

List of available TLS ciphersuites for TLS 1.3 or below.

See the OpenSSL manual page for SSL_CTX_set_ciphersuites for more information on the format of this string.

[default] NULL

6.5.2.3 dh_param_files_length

```
DDS_Long RTITLS_Ciphers::dh_param_files_length
```

Number of DH key files supplied.

Must not be greater than 1 if RTI_SUPPORT_OSSL_DECODER is defined.

[default] 0

6.5.2.4 dh_param_files

```
struct RTITLS_DHParamFile* RTITLS_Ciphers::dh_param_files
```

List of available DH key files.

OpenSSL 3 does not allow generating a file with fewer than 512 bits. According to https://www.openssl.org/docs/man3.0/man3/SSL_CTX_set0_tmp_dh_pkey.html , "Applications may supply their own DH parameters instead of using the built-in values. This approach is discouraged and applications should in preference use the built-in parameter support described above." So setting these files is not recommended.

[default] NULL

6.5.2.5 engine_id

```
char* RTITLS_Ciphers::engine_id
```

ID of OpenSSL cipher engine to request.

[default] NULL

6.6 RTITLS_DHParamFile Struct Reference

Name of a Diffie-Helman (DH) key file and the length of the contained key in bits.

```
#include <rtitls_openssl.h>
```

Data Fields

- char * **file**

  Name of DH key file.
- **DDS_Long bits**

  Length of DH key in bits.

6.6.1 Detailed Description

Name of a Diffie-Helman (DH) key file and the length of the contained key in bits.

6.6.2 Field Documentation

6.6.2.1 file

```
char* RTITLS_DHParamFile::file
```

Name of DH key file.

[default] 0

6.6.2.2 bits

```
DDS_Long RTITLS_DHParamFile::bits
```

Length of DH key in bits.

[default] NULL

6.7 RTITLS_Identity Struct Reference

Set of TLS properties for identity.

```
#include <rtitls_openssl.h>
```

Data Fields

- char * **certificate_chain**

  String containing identifying certificate (in PEM format) or certificate chain (appending intermediate CA certs in order).
- char * **certificate_chain_file**

  File containing identifying certificate (in PEM format) or certificate chain (appending intermediate CA certs in order).
- char * **private_key_password**

  Password for private key.
- char * **private_key**

  String containing private key (in PEM format)
- char * **private_key_file**

  File containing private key (in PEM format)
- char * **rsa_private_key**

  String containing additional RSA private key (in PEM format)
- char * **rsa_private_key_file**

  File containing additional RSA private key (in PEM format)

6.7.1 Detailed Description

Set of TLS properties for identity.

6.7.2 Field Documentation

6.7.2.1 certificate_chain

```
char* RTITLS_Identity::certificate_chain
```

String containing identifying certificate (in PEM format) or certificate chain (appending intermediate CA certs in order).

An identifying certificate is required for secure communication. The string must be sorted starting with the certificate to the highest level (root CA). If this is specified certificate_chain_file must be empty.

[default] NULL

6.7.2.2 certificate_chain_file

```
char* RTITLS_Identity::certificate_chain_file
```

File containing identifying certificate (in PEM format) or certificate chain (appending intermediate CA certs in order).

An identifying certificate is required for secure communication. The file must be sorted starting with the certificate to the highest level (root CA). If this is specified certificate_chain must be empty.

Optionally, a private key may be appended to this file. If no private key option is specified, this file will be used to load a private key.

[default] NULL

6.7.2.3 private_key_password

```
char* RTITLS_Identity::private_key_password
```

Password for private key.

[default] NULL (no password)

6.7.2.4 private_key

```
char* RTITLS_Identity::private_key
```

String containing private key (in PEM format)

At most one of private_key and private_key_file may be specified. If no private key is specified (all values are NULL), the private key will be read from the certificate chain file.

[default] NULL

6.7.2.5 private_key_file

```
char* RTITLS_Identity::private_key_file
```

File containing private key (in PEM format)

At most one of private_key and private_key_file may be specified. If no private key is specified (all values are NULL), the private key will be read from the certificate chain file.

[default] NULL

6.7.2.6 rsa_private_key

```
char* RTITLS_Identity::rsa_private_key
```

String containing additional RSA private key (in PEM format)

For use if both an RSA and non-RSA key are required for the selected cipher. At most one of rsa_private_key and rsa_private_key_file may be specified.

[default] NULL

6.7.2.7 rsa_private_key_file

```
char* RTITLS_Identity::rsa_private_key_file
```

File containing additional RSA private key (in PEM format)

For use if both an RSA and non-RSA key are required for the selected cipher. At most one of rsa_private_key and rsa_private_key_file may be specified.

[default] NULL

6.8 RTITLS_OpenSSL_Configuration Struct Reference

Full set of TLS properties.

`#include <rtitls_openssl.h>`

Data Fields

- struct **RTITLS_Verification verify**
  Verification properties.
- struct **RTITLS_Identity identity**
  Identity properties.
- struct **RTITLS_Ciphers cipher**
  Cipher properties.

6.8.1 Detailed Description

Full set of TLS properties.

Should be initialized with RTITLS_Plugin::RTITLS_OPENSSL_CONFIGURATION_DEFAULT

6.8.2 Field Documentation

6.8.2.1 verify

`struct` **`RTITLS_Verification`** `RTITLS_OpenSSL_Configuration::verify`

Verification properties.

6.8.2.2 identity

`struct` **`RTITLS_Identity`** `RTITLS_OpenSSL_Configuration::identity`

Identity properties.

6.8.2.3 cipher

```
struct RTITLS_Ciphers RTITLS_OpenSSL_Configuration::cipher
```

Cipher properties.

6.9 RTITLS_Verification Struct Reference

Set of TLS properties for certificate authorities (CAs) and verification.

```
#include <rtitls_openssl.h>
```

Data Fields

- char * **ca_file**
  Name of file containing Certificate Authority certificates.
- char * **ca_path**
  Paths to directories containing Certificate Authority certificates.
- char * **ca**
  List of Certificate Authority certificates.
- **DDS_Long verify_depth**
  Maximum certificate chain length for verification.
- **RTITLS_Verify_Callback callback**
  Callback used to verify peer certificates.
- char * **crl_file**
  Name of file containing Certificate Revocation List.

6.9.1 Detailed Description

Set of TLS properties for certificate authorities (CAs) and verification.

6.9.2 Field Documentation

6.9.2.1 ca_file

```
char* RTITLS_Verification::ca_file
```

Name of file containing Certificate Authority certificates.

File should be in PEM format. See the OpenSSL manual page for SSL_load_verify_locations for more information.

At least one of ca_file and ca_path must be specified; both may be specified.

[default] NULL

6.9.2.2 ca_path

```
char* RTITLS_Verification::ca_path
```

Paths to directories containing Certificate Authority certificates.

Files should be in PEM format, and follow the OpenSSL-required naming conventions. See the OpenSSL manual page for SSL_CTX_load_verify_locations for more information.

At least one of ca_file and ca_path must be specified; both may be specified.

[default] NULL

6.9.2.3 ca

```
char* RTITLS_Verification::ca
```

List of Certificate Authority certificates.

Certificates should be in PEM format, and follow the OpenSSL-required naming conventions. See the OpenSSL manual page for X509_STORE_add_cert for more information.

ca_file and ca_path have precedence if specified. If not, ca must be specified.

[default] NULL

6.9.2.4 verify_depth

```
DDS_Long RTITLS_Verification::verify_depth
```

Maximum certificate chain length for verification.

[default] -1 (no limit)

6.9.2.5 callback

```
RTITLS_Verify_Callback RTITLS_Verification::callback
```

Callback used to verify peer certificates.

See the OpenSSL manual page for SSL_set_verify for more information. There are a number of default callbacks included in the Secure Transport. See **RTITLS_default_verify_callback()** (p. 35) , **RTITLS_verbose_verify_callback()** (p. 35) .

[default] NULL (use **RTITLS_default_verify_callback()** (p. 35))

6.9.2.6 crl_file

```
char* RTITLS_Verification::crl_file
```

Name of file containing Certificate Revocation List.

File should be in PEM format.

[default] NULL

Chapter 7

File Documentation

7.1 rtitls_openssl.h

```
1 /*
2  * Copyright (c) 2009-2024 Real-Time Innovations, Inc. All rights reserved.
3  * No duplications, whole or partial, manual or electronic, may be made
4  * without express written permission.  Any such copies, or revisions thereof,
5  * must display this notice unaltered.
6  * This code contains trade secrets of Real-Time Innovations, Inc.
7  */
8
9 #ifndef rtitls_openssl_h
10 #define rtitls_openssl_h
11
12   #ifndef rtitls_dll_h
13     #include "rtitls/rtitls_dll.h"
14   #endif
15
16   #ifndef dds_c_infrastructure_h
17     #include "dds_c/dds_c_infrastructure.h"
18   #endif
19
20 /* if required, include openssl/ssl.h before this file */
21 #ifndef SSL_ERROR_NONE
22   #define X509_STORE_CTX   void
23   #define SSL_CTX          void
24 #endif
25
26 #ifdef __cplusplus
27     extern "C" {
28 #endif
29
30 /* {{{ Doxygen group definitions
31  * ------------------------------------------------------------------------ */
32
33 /*e \defgroup RTITLS_Plugin RTI TLS Support
34   @brief OpenSSL configuration interfaces and definitions.
35   <br>TLS Support is included with some RTI purchases and must be downloaded and installed separately.
36 */
37
38 /* }}} */
39
40 /*i \file
41     \ingroup RTITLS_Plugin
42     \brief TLS Transport OpenSSL configuration interfaces and definitions.
43     <br>TLS Support is included with some RTI purchases and must be downloaded and installed separately.
44 */
45
46 /*e \ingroup RTITLS_Plugin
47     \brief Callback used to verify peer certificates.
48
49     See the OpenSSL manual page for SSL_CTX_set_verify for more information.
50  */
51 typedef int (* RTITLS_Verify_Callback)
```

```
52      (int preverify_ok, X509_STORE_CTX *x509_ctx);
53
54  /*e \ingroup RTITLS_Plugin
55      \brief Set of TLS properties for certificate authorities (CAs) and verification
56   */
57  struct RTITLS_Verification {
58      /*e
59       * \brief Name of file containing Certificate Authority certificates
60       *
61       *  File should be in PEM format. See the OpenSSL manual page for
62       *  SSL_load_verify_locations for more information.
63       *
64       *  At least one of ca_file and ca_path must be specified; both may be
65       *  specified.
66       *
67       * \default NULL
68       */
69      char *ca_file;
70      /*e
71       * \brief Paths to directories containing Certificate Authority certificates
72       *
73       *  Files should be in PEM format, and follow the OpenSSL-required naming
74       *  conventions. See the OpenSSL manual page for SSL_CTX_load_verify_locations
75       *  for more information.
76       *
77       *  At least one of ca_file and ca_path must be specified; both may be
78       *  specified.
79       *
80       * \default NULL
81       */
82      char *ca_path;
83      /*e
84       * \brief List of Certificate Authority certificates
85       *
86       *  Certificates should be in PEM format, and follow the OpenSSL-required naming
87       *  conventions. See the OpenSSL manual page for X509_STORE_add_cert
88       *  for more information.
89       *
90       *  ca_file and ca_path have precedence if specified. If not, ca must be
91       *  specified.
92       *
93       * \default NULL
94       */
95      char *ca;
96
97      /*e
98       * \brief Maximum certificate chain length for verification
99       *
100      *  \default -1 (no limit)
101      */
102     DDS_Long verify_depth;
103     /*e
104      * \brief Callback used to verify peer certificates
105      *
106      *  See the OpenSSL manual page for SSL_set_verify for more information.
107      *  There are a number of default callbacks included in the Secure
108      *  Transport.  See RTITLS_default_verify_callback() ,
109      *  RTITLS_verbose_verify_callback() .
110      *
111      * \default NULL (use RTITLS_default_verify_callback() )
112      */
113     RTITLS_Verify_Callback callback;
114     /*e
115      * \brief Name of file containing Certificate Revocation List
116      *
117      *  File should be in PEM format.
118      *
119      * \default NULL
120      */
121     char *crl_file;
122 };
123
124 /*e \ingroup RTITLS_Plugin
125     \brief Set of TLS properties for identity
126  */
127 struct RTITLS_Identity {
128     /*e \brief String containing identifying certificate (in PEM format) or
129      * certificate chain (appending intermediate CA certs in order).
130      *
131      * An identifying certificate is required for secure communication.
132      * The string must be sorted starting with the certificate to the highest
```

```
133      * level (root CA).  If this is specified certificate_chain_file must be
134      * empty.
135      *
136      * \default NULL
137      */
138     char *certificate_chain;
139
140     /*e \brief File containing identifying certificate (in PEM format) or
141      * certificate chain (appending intermediate CA certs in order).
142      *
143      * An identifying certificate is required for secure communication.
144      * The file must be sorted starting with the certificate to the highest
145      * level (root CA).  If this is specified certificate_chain must be
146      * empty.
147      *
148      * Optionally, a private key may be appended to this file.  If no private
149      * key option is specified, this file will be used to load a private key.
150      *
151      * \default NULL
152      */
153     char *certificate_chain_file;
154
155     /*e @brief Password for private key
156      *
157      * \default NULL (no password)
158      */
159     char *private_key_password;
160
161     /*e \brief String containing private key (in PEM format)
162      *
163      * At most one of private_key and private_key_file may be specified.  If no
164      * private key is specified (all values are NULL), the private key will be read
165      * from the certificate chain file.
166      *
167      * \default NULL
168      */
169     char *private_key;
170
171     /*e \brief File containing private key (in PEM format)
172      *
173      * At most one of private_key and private_key_file may be specified.  If no
174      * private key is specified (all values are NULL), the private key will be read
175      * from the certificate chain file.
176      *
177      * \default NULL
178      */
179     char *private_key_file;
180
181     /*e \brief String containing additional RSA private key (in PEM format)
182      *
183      * For use if both an RSA and non-RSA key are required for the selected cipher.
184      * At most one of rsa_private_key and rsa_private_key_file may be specified.
185      *
186      * \default NULL
187      */
188     char *rsa_private_key;
189
190     /*e \brief File containing additional RSA private key (in PEM format)
191      *
192      * For use if both an RSA and non-RSA key are required for the selected cipher.
193      * At most one of rsa_private_key and rsa_private_key_file may be specified.
194      *
195      * \default NULL
196      */
197     char *rsa_private_key_file;
198 };
199
200 /*e \ingroup RTITLS_Plugin
201     \brief Name of a Diffie-Helman (DH) key file and the length of the
202       contained key in bits
203  */
204 struct RTITLS_DHParamFile {
205     /*e \brief Name of DH key file
206      *
207      * \default 0
208      */
209     char *file;
210
211     /*e \brief Length of DH key in bits
212      *
213      * \default NULL
```

```
214      */
215     DDS_Long bits;
216 };
217
218 /*e \ingroup RTITLS_Plugin
219     \brief Set of TLS properties for cipher configuration
220  */
221 struct RTITLS_Ciphers {
222     /*e \brief List of available TLS ciphers for TLS 1.2 or below
223      *
224      * See the OpenSSL manual page for SSL_set_cipher_list(3) or ciphers(1)
225      * for more information on the format of this string.  Some typical
226      * values are defined: RTITLS_Plugin#RTITLS_CIPHER_LIST_DEFAULT,
227      * RTITLS_Plugin#RTITLS_CIPHER_LIST_ENCRYPT_HIGH, and
228      * RTITLS_Plugin#RTITLS_CIPHER_LIST_UNENCRYPTED.
229      *
230      * \default NULL
231      */
232     char *cipher_list; /* cipher list in OpenSSL format */
233
234     /*e \brief List of available TLS ciphersuites for TLS 1.3 or below
235      *
236      * See the OpenSSL manual page for SSL_CTX_set_ciphersuites
237      * for more information on the format of this string.
238      *
239      * \default NULL
240      */
241     char *ciphersuites; /* ciphersuites in OpenSSL format */
242
243     /*e \brief Number of DH key files supplied
244      *
245      * Must not be greater than 1 if RTI_SUPPORT_OSSL_DECODER is defined.
246      *
247      * \default 0
248      */
249     DDS_Long dh_param_files_length;
250
251     /*e \brief List of available DH key files
252      *
253      * OpenSSL 3 does not allow generating a file with fewer than 512 bits.
254      * According to
255      * https://www.openssl.org/docs/man3.0/man3/SSL_CTX_set0_tmp_dh_pkey.html ,
256      * "Applications may supply their own DH parameters instead of using the
257      * built-in values. This approach is discouraged and applications should in
258      * preference use the built-in parameter support described above." So
259      * setting these files is not recommended.
260      *
261      * \default NULL
262      */
263     struct RTITLS_DHParamFile *dh_param_files;
264
265     /*e \brief ID of OpenSSL cipher engine to request
266      *
267      * \default NULL
268      */
269     char *engine_id;
270
271     /*i \brief pre-load commands: length, name/parameter pairs */
272     DDS_Long engine_pre_cmd_length;
273     const char **engine_pre_cmd_names, **engine_pre_cmd_parameters;
274     /*i \brief post-load commands: length, name/parameter pairs */
275     DDS_Long engine_post_cmd_length;
276     const char **engine_post_cmd_names, **engine_post_cmd_parameters;
277 };
278
279 /*i \ingroup RTITLS_Plugin
280     \brief Set of TLS properties for required renegotiation.
281  */
282 struct RTITLS_Renegotiate {
283     /*i \brief Force renegotiation after connection open for time period
284      *
285      * \default DDS_DURATION_INFINITE
286      */
287     struct DDS_Duration_t force_after_time;
288
289     /*i \brief Force renegotiation after amount of data transferred
290      *
291      * Data sent or received, measured in total unencryted length (0 = no force)
292      *
293      * \default 0
294      */
```

```
295     DDS_UnsignedLong force_after_Kbytes;
296
297     /*i \brief Require renegotiation to complete with this period
298      *
299      * \default DDS_DURATION_INFINITE
300      */
301     struct DDS_Duration_t completion_period;
302 };
303
304 /*i */
305 typedef enum {
306     RTITLS_PROTOCOL_TLS = 0,
307     RTITLS_PROTOCOL_DTLS = 1
308 } RTITLS_Protocol_t;
309
310 /*e \ingroup RTITLS_Plugin
311     \brief Full set of TLS properties
312
313     Should be initialized with RTITLS_Plugin#RTITLS_OPENSSL_CONFIGURATION_DEFAULT
314  */
315 struct RTITLS_OpenSSL_Configuration {
316     /*e \brief Verification properties */
317     struct RTITLS_Verification   verify;
318     /*e \brief Identity properties */
319     struct RTITLS_Identity       identity;
320     /*e \brief Cipher properties */
321     struct RTITLS_Ciphers        cipher;
322     /*i \brief Renegotiation properties */
323     struct RTITLS_Renegotiate    renegotiate;
324 };
325
326 /*e \ingroup RTITLS_Plugin
327   \brief Use this to initialize a RTITLS_Verfication structure.
328
329   \showinitializer
330 */
331 #define RTITLS_VERIFY_DEFAULT { \
332     NULL, NULL, NULL, /* ca_file, ca_path, ca */ \
333     -1, /* verify_depth (no depth limit) */ \
334     NULL, /* callback (use default verify callback) */ \
335     NULL /* crl_file */ }
336
337 /*e \ingroup RTITLS_Plugin
338   \brief Use this to initialize a RTITLS_Identity structure.
339
340   \showinitializer
341 */
342 #define RTITLS_IDENTITY_DEFAULT { \
343     NULL, /* certificate_chain */ \
344     NULL, /* certificate_chain_file */ \
345     NULL, /* private_key_password */ \
346     NULL, /* private_key */ \
347     NULL, /* private_key_file */ \
348     NULL, /* rsa_private_key */ \
349     NULL /* rsa_private_key_file */ }
350
351 /*e \ingroup RTITLS_Plugin
352   \brief Cipher list string for default channel (encrypted) */
353 #define RTITLS_CIPHER_LIST_DEFAULT          "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH"
354
355 /*e \ingroup RTITLS_Plugin
356   \brief Cipher list string for default channel (encrypted, no low-strength) */
357 #define RTITLS_CIPHER_LIST_ENCRYPT_HIGH     "AES:ALL:!aNULL:!eNULL:!LOW:!EXP:+RC4:@STRENGTH"
358
359 /*e \ingroup RTITLS_Plugin
360   \brief Cipher list string for authentication-only channel (no encryption) */
361 #define RTITLS_CIPHER_LIST_UNENCRYPTED      "aNULL"
362
363 /*e \ingroup RTITLS_Plugin
364   \brief Use this to initialize a RTITLS_Ciphers structure.
365
366   \showinitializer
367 */
368 #define RTITLS_CIPHER_DEFAULT { \
369     NULL, /* cipher_list */ \
370     NULL, /* ciphersuites */ \
371     0, NULL, /* dh_param_files_length, dh_param_files (no DH params) */ \
372     NULL, /* engine_id (no engine) */ \
373     0, NULL, NULL, /* engine_pre_cmd_length, engine_pre_cmd_names, engine_pre_cmd_parameters */ \
374     0, NULL, NULL /* engine_post_cmd_length, engine_post_cmd_names, engeine_post_cmd_parameters */ }
375
```

```
376 /*i \ingroup RTITLS_Plugin
377   \brief Use this to initialize a RTITLS_Renegotiate structure.
378
379   \showinitializer
380 */
381 #define RTITLS_RENEGOTIATE_DEFAULT { \
382     DDS_DURATION_INFINITE_VALUE, /* force_after_time */ \
383     0, /* force_after_Kbytes */ \
384     DDS_DURATION_INFINITE_VALUE /* completion_period */ }
385
386 /*e \ingroup RTITLS_Plugin
387   \brief Use this to initialize a RTITLS_OpenSSL_Configuration structure.
388
389   \showinitializer
390 */
391 #define RTITLS_OPENSSL_CONFIGURATION_DEFAULT { \
392     RTITLS_VERIFY_DEFAULT, /* verify */ \
393     RTITLS_IDENTITY_DEFAULT, /* identity */ \
394     RTITLS_CIPHER_DEFAULT, /* cipher */ \
395     RTITLS_RENEGOTIATE_DEFAULT /* renegotiate */ }
396
397 /*e \ingroup RTITLS_Plugin
398   \brief clean up OpenSSL resources for current thread (call before exit)
399 */
400 extern RTITLSDllExport
401 void RTITLS_thread_exit(void);
402
403 /*e \ingroup RTITLS_Plugin
404   \brief Default verify callback: log errors when verification fails
405
406   See the OpenSSL manual page for SSL_CTX_set_verify for more information.
407 */
408 extern int RTITLS_default_verify_callback(int ok, X509_STORE_CTX *store);
409
410 /*e \ingroup RTITLS_Plugin
411   \brief Verbose verify callback: log information about successful verification
412     as well as errors when verification fails
413
414   See the OpenSSL manual page for SSL_CTX_set_verify for more information.
415 */
416 extern int RTITLS_verbose_verify_callback(int ok, X509_STORE_CTX *store);
417
418 /*
419  * tls is only compiled with OpenSSL support.
420  * In the future this may change to support WolfSSL.
421  */
422 #if defined(RTI_OPENSSL_ARCHITECTURE) || defined(RTI_OPENSSL3_ARCHITECTURE)
423   /* methods below only supported on TLS architectures */
424
425 /*i \brief verify protocol configuration struct */
426 extern RTITLSDllExport
427 int RTITLS_configuration_verify(
428     const struct RTITLS_OpenSSL_Configuration *config);
429
430 /*i \brief configure protocol, identity based on configuration struct */
431 extern RTITLSDllExport
432 SSL_CTX *RTITLS_context_init(
433     RTITLS_Protocol_t protocol,
434     const struct RTITLS_OpenSSL_Configuration *config);
435
436 /*i \brief clean up OpenSSL resources for context */
437 extern RTITLSDllExport
438 void RTITLS_context_done(SSL_CTX *context);
439
440 /* ========================================================== */
441 #endif /* RTI_OPENSSL_ARCHITECTURE || RTI_OPENSSL3_ARCHITECTURE */
442
443 #ifdef __cplusplus
444     }   /* extern "C" */
445 #endif
446
447 #endif /* rtitls_openssl_h */
```

Index

assertions_per_lease_duration
    NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t, 74

bind_interface_address
    NDDS_Transport_TCPv4_Property_t, 81
bits
    RTITLS_DHParamFile, 94

C Example of TCP Transport with PropertiesQosPolicy Configuration, 36
ca
    RTITLS_Verification, 99
ca_file
    RTITLS_Verification, 98
ca_path
    RTITLS_Verification, 98
callback
    RTITLS_Verification, 99
certificate_chain
    RTITLS_Identity, 95
certificate_chain_file
    RTITLS_Identity, 95
cipher
    RTITLS_OpenSSL_Configuration, 97
cipher_list
    RTITLS_Ciphers, 92
ciphersuites
    RTITLS_Ciphers, 92
client_connection_negotiation_timeout
    NDDS_Transport_TCPv4_Property_t, 89
concurrency_value
    NDDS_Transport_TCPv4_WindowsIOCPSettings_t, 91
Configure TCP Transport with Property QoS Policy, 22
Connection Endpoint Abstractions, 9
connection_liveliness_settings
    NDDS_Transport_TCPv4_Property_t, 86
control_attribute_allocation
    NDDS_Transport_TCPv4_Property_t, 83
control_buffer_allocation
    NDDS_Transport_TCPv4_Property_t, 83
control_message_allocation
    NDDS_Transport_TCPv4_Property_t, 83
crl_file
    RTITLS_Verification, 99

dh_param_files
    RTITLS_Ciphers, 93
dh_param_files_length
    RTITLS_Ciphers, 93
disable_interface_tracking
    NDDS_Transport_TCPv4_Property_t, 90
disable_nagle
    NDDS_Transport_TCPv4_Property_t, 86

enable
    NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t, 73
enable_keep_alive
    NDDS_Transport_TCPv4_Property_t, 85
engine_id
    RTITLS_Ciphers, 93
event_thread_settings
    NDDS_Transport_TCPv4_Property_t, 86

file
    RTITLS_DHParamFile, 94
force_asynchronous_send
    NDDS_Transport_TCPv4_Property_t, 83
force_crc_check
    NDDS_Transport_TCPv4_Property_t, 88
force_interface_poll_detection
    NDDS_Transport_TCPv4_Property_t, 80

HelloCommon.h, 40
HelloWorld.c, 46
HelloWorld.idl, 39
HelloWorld_publisher.c, 51
HelloWorld_subscriber.c, 54
HelloWorldPlugin.c, 58
HelloWorldSupport.c, 69

identity
    RTITLS_OpenSSL_Configuration, 97
ignore_loopback_interface
    NDDS_Transport_TCPv4_Property_t, 78
ignore_nonrunning_interfaces
    NDDS_Transport_TCPv4_Property_t, 78
initial_handshake_timeout
    NDDS_Transport_TCPv4_Property_t, 90
interface_poll_period
    NDDS_Transport_TCPv4_Property_t, 80

keep_alive_interval
NDDS_Transport_TCPv4_Property_t, 85
keep_alive_retry_count
NDDS_Transport_TCPv4_Property_t, 85
keep_alive_time
NDDS_Transport_TCPv4_Property_t, 85

lease_duration
NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t, 73
logging_verbosity_bitmap
NDDS_Transport_TCPv4_Property_t, 86

max_packet_size
NDDS_Transport_TCPv4_Property_t, 84

NDDS_TRANSPORT_CLASSNAME_TCPV4_LAN
TCP Transport Plugin API, 12
NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN
TCP Transport Plugin API, 12
NDDS_TRANSPORT_TCPV4_ADDRESS_BIT_COUNT
TCP Transport Plugin API, 13
NDDS_TRANSPORT_TCPV4_ADDRESS_LOCALHOST
TCP Transport Plugin API, 17
NDDS_TRANSPORT_TCPV4_CONNECTION_LIVELINESS_SETTINGS_DEFAULT
TCP Transport Plugin API, 14
NDDS_Transport_TCPv4_ConnectionLivelinessSettings_t, 73
assertions_per_lease_duration, 74
enable, 73
lease_duration, 73
NDDS_TRANSPORT_TCPV4_CONTROL_BUFFER_POOL_GROWTH_POLICY_DEFAULT
TCP Transport Plugin API, 14
NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_ATTRIBUTE_FACTORY_GROWTH_POLICY_DEFAULT
TCP Transport Plugin API, 14
NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_FACTORY_GROWTH_POLICY_DEFAULT
TCP Transport Plugin API, 14
NDDS_Transport_TCPv4_create
TCP Transport Plugin API, 20
NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER
TCP Transport Plugin API, 12
NDDS_TRANSPORT_TCPV4_EVENT_THREAD_SETTINGS_DEFAULT
TCP Transport Plugin API, 14
NDDS_Transport_TCPv4_EventThreadSettings_t, 74
options, 75
priority, 74
stack_size, 74
NDDS_Transport_TCPv4_get_build_version_string
Version, 22
NDDS_Transport_TCPv4_get_library_version
Version, 22
NDDS_Transport_TCPv4_LibraryVersion
Version, 21
NDDS_TRANSPORT_TCPV4_MESSAGE_SIZE_MAX_DEFAULT
TCP Transport Plugin API, 12
NDDS_Transport_TCPv4_new
TCP Transport Plugin API, 19
NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_CODE
TCP Transport Plugin API, 18
NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_CODE
TCP Transport Plugin API, 18
NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_DEST
TCP Transport Plugin API, 18
NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_UNKN
TCP Transport Plugin API, 18
NDDS_Transport_TCPv4_OnConnectionEstablishedCallback
TCP Transport Plugin API, 17
NDDS_Transport_TCPv4_OnConnectionLost_ReasonCode_t
TCP Transport Plugin API, 18
NDDS_Transport_TCPv4_OnConnectionLostCallback
TCP Transport Plugin API, 18
NDDS_Transport_TCPv4_Plugin_stringToTransportAddress
TCP Transport Plugin API, 20
NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT
TCP Transport Plugin API, 17
NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT_LAN
TCP Transport Plugin API, 15
NDDS_Transport_TCPv4_Property_t, 75
bind_interface_address, 81
client_connection_negotiation_timeout, 89
connection_liveliness_settings, 86
control_attribute_allocation, 83
control_buffer_allocation, 83
control_message_allocation, 83
disable_interface_tracking, 90
disable_nagle, 86
enable_keep_alive, 85
event_thread_settings, 86
force_asynchronous_send, 83
force_crc_check, 88
force_interface_poll_detection, 80
ignore_loopback_interface, 78
ignore_nonrunning_interfaces, 78
initial_handshake_timeout, 90
interface_poll_period, 80
keep_alive_interval, 85
keep_alive_retry_count, 85
keep_alive_time, 85
logging_verbosity_bitmap, 86
max_packet_size, 84
on_connection_established, 84
on_connection_lost, 84
outstanding_connection_cookies, 88
outstanding_connection_cookies_life_span, 88
parent, 77
public_address, 81
read_buffer_allocation, 82
recv_socket_buffer_size, 78
security_logging_verbosity_bitmap, 87

send_crc, 88
send_max_wait_sec, 89
send_socket_buffer_size, 78
server_bind_port, 81
server_connection_negotiation_timeout, 89
server_socket_backlog, 81
shared_write_buffer_allocation, 82
socket_monitoring_kind, 87
tls, 90
transport_priority_mapping_high, 80
transport_priority_mapping_low, 79
transport_priority_mask, 79
user_timeout, 86
windows_iocp_settings, 88
write_buffer_allocation, 82
NDDS_TRANSPORT_TCPV4_READ_BUFFER_POOL_GROWTH_POLICY_DEFAULT
TCP Transport Plugin API, 13
NDDS_TRANSPORT_TCPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT
TCP Transport Plugin API, 13
NDDS_TRANSPORT_TCPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT
TCP Transport Plugin API, 13
NDDS_TRANSPORT_TCPV4_SERVER_SOCKET_BACKLOG_DEFAULT
TCP Transport Plugin API, 13
NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT
TCP Transport Plugin API, 12
NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_DEFAULT
TCP Transport Plugin API, 15
NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_POLL
TCP Transport Plugin API, 19
NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_SELECT
TCP Transport Plugin API, 19
NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_WINDOWS_IOCP
TCP Transport Plugin API, 19
NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_WINDOWS_WAITFORMULTIPLEOBJECTS
TCP Transport Plugin API, 19
NDDS_Transport_TCPv4_SocketMonitoringKind_t
TCP Transport Plugin API, 19
NDDS_TRANSPORT_TCPV4_WINDOWS_IOCP_SETTINGS_DEFAULT
TCP Transport Plugin API, 15
NDDS_Transport_TCPv4_WindowsIOCPSettings_t, 91
concurrency_value, 91
thread_pool_size, 91
NDDS_TRANSPORT_TCPV4_WRITE_BUFFER_POOL_GROWTH_POLICY_DEFAULT
TCP Transport Plugin API, 13

on_connection_established
NDDS_Transport_TCPv4_Property_t, 84
on_connection_lost
NDDS_Transport_TCPv4_Property_t, 84
options
NDDS_Transport_TCPv4_EventThreadSettings_t, 75
outstanding_connection_cookies
NDDS_Transport_TCPv4_Property_t, 88
outstanding_connection_cookies_life_span
NDDS_Transport_TCPv4_Property_t, 88

parent
NDDS_Transport_TCPv4_Property_t, 77
priority
NDDS_Transport_TCPv4_EventThreadSettings_t, 74
private_key
RTITLS_Identity, 96
private_key_file
RTITLS_Identity, 96
private_key_password
RTITLS_Identity, 95
public_address
NDDS_Transport_TCPv4_Property_t, 81

read_buffer_allocation
NDDS_Transport_TCPv4_Property_t, 82
recv_socket_buffer_size
NDDS_Transport_TCPv4_Property_t, 78
rsa_private_key
RTITLS_Identity, 96
rsa_private_key_file
RTITLS_Identity, 96
RTI TLS Support, 32
RTITLS_CIPHER_DEFAULT, 34
RTITLS_CIPHER_LIST_DEFAULT, 33
RTITLS_CIPHER_LIST_ENCRYPT_HIGH, 34
RTITLS_CIPHER_LIST_UNENCRYPTED, 34
RTITLS_default_verify_callback, 35
RTITLS_IDENTITY_DEFAULT, 33
RTITLS_OPENSSL_CONFIGURATION_DEFAULT, 34
RTITLS_thread_exit, 35
RTITLS_verbose_verify_callback, 35
RTITLS_Verify_Callback, 35
RTITLS_VERIFY_DEFAULT, 33
RTITLS_CIPHER_DEFAULT
RTI TLS Support, 34
RTITLS_CIPHER_LIST_DEFAULT
RTI TLS Support, 33
RTITLS_CIPHER_LIST_ENCRYPT_HIGH
RTI TLS Support, 34
RTITLS_CIPHER_LIST_UNENCRYPTED
RTI TLS Support, 34
RTITLS_Ciphers, 92
cipher_list, 92
ciphersuites, 92
dh_param_files, 93
dh_param_files_length, 93
engine_id, 93
RTITLS_default_verify_callback
RTI TLS Support, 35
RTITLS_DHParamFile, 93

bits, 94
file, 94
RTITLS_Identity, 94
certificate_chain, 95
certificate_chain_file, 95
private_key, 96
private_key_file, 96
private_key_password, 95
rsa_private_key, 96
rsa_private_key_file, 96
RTITLS_IDENTITY_DEFAULT
RTI TLS Support, 33
rtitls_openssl.h, 101
RTITLS_OpenSSL_Configuration, 97
cipher, 97
identity, 97
verify, 97
RTITLS_OPENSSL_CONFIGURATION_DEFAULT
RTI TLS Support, 34
RTITLS_thread_exit
RTI TLS Support, 35
RTITLS_verbose_verify_callback
RTI TLS Support, 35
RTITLS_Verification, 98
ca, 99
ca_file, 98
ca_path, 98
callback, 99
crl_file, 99
verify_depth, 99
RTITLS_Verify_Callback
RTI TLS Support, 35
RTITLS_VERIFY_DEFAULT
RTI TLS Support, 33

security_logging_verbosity_bitmap
NDDS_Transport_TCPv4_Property_t, 87
send_crc
NDDS_Transport_TCPv4_Property_t, 88
send_max_wait_sec
NDDS_Transport_TCPv4_Property_t, 89
send_socket_buffer_size
NDDS_Transport_TCPv4_Property_t, 78
server_bind_port
NDDS_Transport_TCPv4_Property_t, 81
server_connection_negotiation_timeout
NDDS_Transport_TCPv4_Property_t, 89
server_socket_backlog
NDDS_Transport_TCPv4_Property_t, 81
shared_write_buffer_allocation
NDDS_Transport_TCPv4_Property_t, 82
Socket Utilities, 9
socket_monitoring_kind
NDDS_Transport_TCPv4_Property_t, 87
stack_size
NDDS_Transport_TCPv4_EventThreadSettings_t, 74

TCP Transport Examples, 31
TCP Transport Initial Peers, 22
TCP Transport Plugin API, 9
NDDS_TRANSPORT_CLASSNAME_TCPV4_LAN, 12
NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN, 12
NDDS_TRANSPORT_TCPV4_ADDRESS_BIT_COUNT, 13
NDDS_TRANSPORT_TCPV4_ADDRESS_LOCALHOST, 17
NDDS_TRANSPORT_TCPV4_CONNECTION_LIVELINESS_SETTING 14
NDDS_TRANSPORT_TCPV4_CONTROL_BUFFER_POOL_GROWTH 14
NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_ATTRIBUTE_F 14
NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_FACTORY_GR 14
NDDS_Transport_TCPv4_create, 20
NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER, 12
NDDS_TRANSPORT_TCPV4_EVENT_THREAD_SETTINGS_DEFAU 14
NDDS_TRANSPORT_TCPV4_MESSAGE_SIZE_MAX_DEFAULT, 12
NDDS_Transport_TCPv4_new, 19
NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_ 18
NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_ 18
NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_ 18
NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_ 18
NDDS_Transport_TCPv4_OnConnectionEstablishedCallback, 17
NDDS_Transport_TCPv4_OnConnectionLost_ReasonCode_t, 18
NDDS_Transport_TCPv4_OnConnectionLostCallback, 18
NDDS_Transport_TCPv4_Plugin_stringToTransportAddress, 20
NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT, 17
NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT_LAN, 15
NDDS_TRANSPORT_TCPV4_READ_BUFFER_POOL_GROWTH_PO 13
NDDS_TRANSPORT_TCPV4_RECV_SOCKET_BUFFER_SIZE_DEF

13
NDDS_TRANSPORT_TCPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT,
13
NDDS_TRANSPORT_TCPV4_SERVER_SOCKET_BACKLOG_DEFAULT,
13
NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT,
12
NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_DEFAULT,
15
NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_POLL,
19
NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_SELECT,
19
NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_WINDOWS_IOCP,
19
NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_WINDOWS_WAITFORMULTIPLEOBJECTS,
19
NDDS_Transport_TCPv4_SocketMonitoringKind_t,
19
NDDS_TRANSPORT_TCPV4_WINDOWS_IOCP_SETTINGS_DEFAULT,
15
NDDS_TRANSPORT_TCPV4_WRITE_BUFFER_POOL_GROWTH_POLICY_DEFAULT,
13
thread_pool_size
NDDS_Transport_TCPv4_WindowsIOCPSettings_t,
91
tls
NDDS_Transport_TCPv4_Property_t, 90
transport_priority_mapping_high
NDDS_Transport_TCPv4_Property_t, 80
transport_priority_mapping_low
NDDS_Transport_TCPv4_Property_t, 79
transport_priority_mask
NDDS_Transport_TCPv4_Property_t, 79

user_timeout
NDDS_Transport_TCPv4_Property_t, 86

verify
RTITLS_OpenSSL_Configuration, 97
verify_depth
RTITLS_Verification, 99
Version, 21
NDDS_Transport_TCPv4_get_build_version_string,
22
NDDS_Transport_TCPv4_get_library_version, 22
NDDS_Transport_TCPv4_LibraryVersion, 21

windows_iocp_settings
NDDS_Transport_TCPv4_Property_t, 88
write_buffer_allocation
NDDS_Transport_TCPv4_Property_t, 82