RTI Cloud Discovery Service

Version 7.7.0

1 RTI Cloud Discovery Service **1**

1.1 Feedback and Support for this Release. 1

2 Module Index **3**

2.1 Modules 3

3 Namespace Index **5**

3.1 Namespace List 5

4 Data Structure Index **7**

4.1 Data Structures 7

5 File Index **9**

5.1 File List 9

6 Module Documentation **11**

6.1 Logging 11

6.1.1 Detailed Description 11

6.2 Library API 11

6.2.1 Detailed Description 12

7 Namespace Documentation **15**

7.1 rti Namespace Reference 15

7.1.1 Detailed Description 15

8 Data Structure Documentation **17**

8.1 rti::cds::DomainListProperty Class Reference 17

8.1.1 Detailed Description 17

8.1.2 Member Function Documentation 17

8.1.2.1 allow_domain_ids() [1/2] 18

8.1.2.2 allow_domain_ids() [2/2] 18

8.1.2.3 deny_domain_ids() [1/2] 18

8.1.2.4 deny_domain_ids() [2/2] 18

8.2 rti::cds::Logger Class Reference 19

8.2.1 Detailed Description 19

8.2.2 Member Function Documentation 19

8.2.2.1 service_verbosity() [1/2] 19

8.2.2.2 service_verbosity() [2/2] 20

8.2.2.3 error() [1/2] 20

8.2.2.4 error() [2/2] 20

8.2.2.5 warn() [1/2] 21

8.2.2.6 warn() [2/2] 21

8.2.2.7 local() [1/2] . . . 21
8.2.2.8 local() [2/2] . . . 22
8.2.2.9 remote() [1/2] . . . 22
8.2.2.10 remote() [2/2] . . . 22
8.2.2.11 debug() [1/2] . . . 23
8.2.2.12 debug() [2/2] . . . 23
8.3 rti::cds::Service Class Reference . . . 23
8.3.1 Detailed Description . . . 24
8.3.2 Constructor & Destructor Documentation . . . 24
8.3.2.1 Service() . . . 24
8.3.3 Member Function Documentation . . . 24
8.3.3.1 start() . . . 24
8.3.3.2 stop() . . . 24
8.3.3.3 is_started() . . . 25
8.3.3.4 update_rtps_psk_secret_passphrase() . . . 25
8.3.3.5 finalize_globals() . . . 25
8.4 rti::cds::ServiceProperty Class Reference . . . 25
8.4.1 Detailed Description . . . 27
8.4.2 Constructor & Destructor Documentation . . . 27
8.4.2.1 ServiceProperty() . . . 27
8.4.3 Member Function Documentation . . . 27
8.4.3.1 cfg_file() [1/2] . . . 27
8.4.3.2 cfg_file() [2/2] . . . 28
8.4.3.3 cfg_strings() [1/2] . . . 28
8.4.3.4 cfg_strings() [2/2] . . . 28
8.4.3.5 service_name() [1/2] . . . 28
8.4.3.6 service_name() [2/2] . . . 29
8.4.3.7 application_name() [1/2] . . . 29
8.4.3.8 application_name() [2/2] . . . 29
8.4.3.9 enforce_xsd_validation() [1/2] . . . 29
8.4.3.10 enforce_xsd_validation() [2/2] . . . 30
8.4.3.11 skip_default_files() [1/2] . . . 30
8.4.3.12 skip_default_files() [2/2] . . . 30
8.4.3.13 transport_selection() [1/2] . . . 30
8.4.3.14 transport_selection() [2/2] . . . 31
8.4.3.15 domain_list() [1/2] . . . 31
8.4.3.16 domain_list() [2/2] . . . 31
8.4.3.17 enable_administration() [1/2] . . . 31
8.4.3.18 enable_administration() [2/2] . . . 32

8.4.3.19 administration_domain_id() [1/2] . . . 32
8.4.3.20 administration_domain_id() [2/2] . . . 32
8.4.3.21 enable_monitoring() [1/2] . . . 32
8.4.3.22 enable_monitoring() [2/2] . . . 33
8.4.3.23 monitoring_domain_id() [1/2] . . . 33
8.4.3.24 monitoring_domain_id() [2/2] . . . 33
8.4.3.25 license_file_name() [1/2] . . . 33
8.4.3.26 license_file_name() [2/2] . . . 34
8.4.3.27 user_environment() [1/2] . . . 34
8.4.3.28 user_environment() [2/2] . . . 34
8.5 rti::cds::TransportUnicastProperty Class Reference . . . 34
8.5.1 Detailed Description . . . 35
8.5.2 Member Function Documentation . . . 35
8.5.2.1 alias() [1/2] . . . 35
8.5.2.2 alias() [2/2] . . . 35
8.5.2.3 receive_port() [1/2] . . . 36
8.5.2.4 receive_port() [2/2] . . . 36

9 File Documentation **37**
9.1 CloudDiscoveryServiceImpl.hpp . . . 37
9.2 Logger.hpp . . . 38
9.3 PropertySet.hpp . . . 39
9.4 Service.hpp . . . 40
9.5 ServiceProperty.hpp . . . 42

Index **49**

Chapter 1

RTI Cloud Discovery Service

RTI Cloud Discovery Service is a stand-alone application that bootstraps discovery for RTI Connext DDS applications in dynamic environments where UDP/IP multicast is not available. This is typical of wide-area networks or some cloud-based environments where the routers and switches may disable IP multicast forwarding. RTI Cloud Discovery Service also works in conjunction with the RTI Real-Time WAN Transport to provide peer-to-peer communication between DomainParticipants situated behind Network Address Translators (NATs).

You can learn how to use RTI Cloud Discovery Service through the `RTI Cloud Discovery Service User's Manual`.

RTI Cloud Discovery Service can be run as a standalone application, or linked into your application.

A reference to the Service (Library) API can be found here:

- **Library API** (p. 11)

1.1 Feedback and Support for this Release.

For more information, visit our knowledge base, accessible from `https://community.rti.com/` to see sample code, general information on RTI Connext, performance information, troubleshooting tips, and technical details.

By its very nature, the knowledge base is continuously evolving and improving. We hope that you will find it helpful. If there are questions that you would like to see addressed or comments you would like to share, please send email to `support@rti.com`. We can only guarantee a response for customers with a current maintenance contract or subscription. To purchase a maintenance contract or subscription, contact your local RTI representative (see `http://www.rti.com/company/contact.html`), send an email request to `sales@rti.com`, or call +1 (408) 990-7400.

Please do not hesitate to contact RTI with questions or comments about this release. We welcome any input on how to improve RTI Cloud Discovery Service and RTI Connext to suit your needs.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Logging 11
Library API 11

Chapter 3

Namespace Index

3.1 Namespace List

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

rti 15

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

rti::cds::DomainListProperty
Specifies the list of domain IDs to allow or deny. Overrides XML configuration if specified 17
rti::cds::Logger
The singleton type used to configure RTI Cloud Discovery **Service** (p. 23) verbosity 19
rti::cds::Service
The RTI Cloud Discovery **Service** (p. 23) 23
rti::cds::ServiceProperty
Configuration properties for a RTI Cloud Discovery **Service** (p. 23) object 25
rti::cds::TransportUnicastProperty
Specifies the alias name of the transport to be used by RTI Cloud Discovery **Service** (p. 23) for sending or receiving announcements. It also indicates the port where incoming participant announcements are received. Overrides XML configuration if specified 34

Chapter 5

File Index

5.1 File List

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

CloudDiscoveryServiceImpl.hpp 37
Logger.hpp 38
PropertySet.hpp 39
Service.hpp 40
ServiceProperty.hpp 42

Chapter 6

Module Documentation

6.1 Logging

Provides control over the logging facilities for RTI Cloud Discovery **Service** (p. 23).

Data Structures

- class **rti::cds::Logger**

  *The singleton type used to configure RTI Cloud Discovery **Service** (p. 23) verbosity.*

6.1.1 Detailed Description

Provides control over the logging facilities for RTI Cloud Discovery **Service** (p. 23).

6.2 Library API

RTI Cloud Discovery **Service** (p. 23) can be deployed as a native library linked into your application in select architectures.

Data Structures

- class **rti::cds::Service**

  *The RTI Cloud Discovery **Service** (p. 23).*
- class **rti::cds::DomainListProperty**

  Specifies the list of domain IDs to allow or deny. Overrides XML configuration if specified.
- class **rti::cds::TransportUnicastProperty**

  *Specifies the alias name of the transport to be used by RTI Cloud Discovery **Service** (p. 23) for sending or receiving announcements. It also indicates the port where incoming participant announcements are received. Overrides XML configuration if specified.*
- class **rti::cds::ServiceProperty**

  *Configuration properties for a RTI Cloud Discovery **Service** (p. 23) object.*

6.2.1 Detailed Description

RTI Cloud Discovery **Service** (p. 23) can be deployed as a native library linked into your application in select architectures.

This API allows you to create, configure and start RTI Cloud Discovery **Service** (p. 23) instances from your application.

The following code shows the typical use of the API:

```
int main ()
{
    rti::cds::Service service(
            rti::cds::ServiceProperty()
            .cfg_file("MyCloudDiscoveryService.xml")
            .service_name("MyCloudDiscoveryService"));
    ...
    service.start();
    while (keep_running) {
        sleep();
        ...
    }
    return 0;
}
```

Instead of a file, you can use XML strings to configure RTI Cloud Discovery **Service** (p. 23). See **ServiceProperty** (p. 25) for more information.

To build your application you need to link with the RTI Cloud Discovery **Service** (p. 23) native library in <**RTI Connext home**>/**bin**/<**architecture**>/ .

An example is provided at **https://github.com/rticommunity/rticonnextdds-examples/tree/master/exampl _discovery_service/library_api**.

	Linux Systems	Windows Systems
Link Libraries (Compile Time)	librticlouddiscoveryservice.so	rticlouddiscoveryservice.dll
Shared Libraries (Run Time)	librtiroutingservice.so	rtiroutingservice.dll
	librtidlc.so	rtidlc.dll
	librticonnextmsgc.so	rticonnextmsgc.dll
	libnddsmetp.so	nddsmetp.dll
	librtiapputilsc.so	rtiapputilsc.dll
	librtixml2.so	rtixml2.dll
	libnddscpp2.so	nddscpp2.dll
	libnddsc.so	nddsc.dll
	libnddscore.so	nddscore.dll
Static Libraries	librticlouddiscoveryservicez.a	rticlouddiscoveryservicez.lib
	librtiroutingservicez.a	rtiroutingservicez.lib
	librtidlcz.a	rtidlcz.lib
	librticonnextmsgcz.a	rticonnextmsgcz.lib
	libnddsmetpz.a	nddsmetpz.lib
	librtiapputilscz.a	rtiapputilscz.lib
	librtixml2z.a	rtixml2z.lib
	libnddscpp2z.a	nddscpp2z.lib
	libnddscz.a	nddscz.lib
	libnddscorez.a	nddscorez.lib
Headers	rti/cds/CloudDiscoveryService.hpp	

6.2.1.0.1 Development Requirements **NOTE:** If you are using debug libraries, remember to add the 'd' suffix to the library name. For shared libraries on macOS systems, the library name is the same as that for Linux except the extension is '.dylib' instead of '.so'.

Chapter 7

Namespace Documentation

7.1 rti Namespace Reference

7.1.1 Detailed Description

(c) Copyright, Real-Time Innovations, 2020. All rights reserved.

No duplications, whole or partial, manual or electronic, may be made without express written permission. Any such copies, or revisions thereof, must display this notice unaltered. This code contains trade secrets of Real-Time Innovations, Inc.

(c) Copyright, Real-Time Innovations, 2020. All rights reserved.

No duplications, whole or partial, manual or electronic, may be made without express written permission. Any such copies, or revisions thereof, must display this notice unaltered. This code contains trade secrets of Real-Time Innovations, Inc. IMPORTANT: macros.hpp must be the first RTI header included in every header file so that symbols are exported correctly on Windows

Chapter 8

Data Structure Documentation

8.1 rti::cds::DomainListProperty Class Reference

Specifies the list of domain IDs to allow or deny. Overrides XML configuration if specified.

`#include <ServiceProperty.hpp>`

Public Member Functions

- void **allow_domain_ids** (std::string allow_domain_ids)

  *The list of domain IDs where RTI Cloud Discovery **Service** (p. 23) processes the incoming participant announcements.*
- std::string & **allow_domain_ids** ()

  Getter (see setter with the same name)
- void **deny_domain_ids** (std::string deny_domain_ids)

  *The list of domain IDs to deny or ignore. Overrides **allow_domain_ids** (p. 17).*
- std::string & **deny_domain_ids** ()

  Getter (see setter with the same name)

8.1.1 Detailed Description

Specifies the list of domain IDs to allow or deny. Overrides XML configuration if specified.

8.1.2 Member Function Documentation

8.1.2.1 allow_domain_ids() [1/2]

```
void rti::cds::DomainListProperty::allow_domain_ids (
            std::string allow_domain_ids )  [inline]
```

The list of domain IDs where RTI Cloud Discovery **Service** (p. 23) processes the incoming participant announcements.

[default] Empty string. Use value from XML configuration.

References **allow_domain_ids()**.

8.1.2.2 allow_domain_ids() [2/2]

```
std::string & rti::cds::DomainListProperty::allow_domain_ids ( )  [inline]
```

Getter (see setter with the same name)

Referenced by **allow_domain_ids()**.

8.1.2.3 deny_domain_ids() [1/2]

```
void rti::cds::DomainListProperty::deny_domain_ids (
            std::string deny_domain_ids )  [inline]
```

The list of domain IDs to deny or ignore. Overrides **allow_domain_ids** (p. 17).

[default] Empty string. Use value from XML configuration.

References **deny_domain_ids()**.

8.1.2.4 deny_domain_ids() [2/2]

```
std::string & rti::cds::DomainListProperty::deny_domain_ids ( )  [inline]
```

Getter (see setter with the same name)

Referenced by **deny_domain_ids()**.

8.2 rti::cds::Logger Class Reference

The singleton type used to configure RTI Cloud Discovery **Service** (p. 23) verbosity.

`#include <Logger.hpp>`

Public Member Functions

- void **service_verbosity** (rti::config::Verbosity verbosity)
  *Sets the verbosity for the log messages generated at the RTI Cloud Discovery **Service** (p. 23) level.*
- rti::config::Verbosity **service_verbosity** ()
  Getter for the same attribute.
- void **error** (const std::string &msg)
  Logs a message with EXCEPTION level.
- void **error** (const char *msg)
  *Overload of **error(const std::string& msg)** (p. 20)*
- void **warn** (const std::string &msg)
  Logs as message with WARNING level.
- void **warn** (const char *msg)
  *Overload of **warn(const std::string& msg)** (p. 20)*
- void **local** (const std::string &msg)
  Logs as message with WARNING level.
- void **local** (const char *msg)
  *Overload of **local(const std::string& msg)** (p. 21)*
- void **remote** (const std::string &msg)
  Logs a message with WARNING level.
- void **remote** (const char *msg)
  *Overload of **remote(const std::string& msg)** (p. 22)*
- void **debug** (const std::string &msg)
  Logs as message with WARNING level.
- void **debug** (const char *msg)
  *Overload of **debug(const std::string& msg)** (p. 22)*

8.2.1 Detailed Description

The singleton type used to configure RTI Cloud Discovery **Service** (p. 23) verbosity.

For configuring other aspects such as output file, print format or RTI Connext specific logging, use rti::config::Logger.

8.2.2 Member Function Documentation

8.2.2.1 service_verbosity() [1/2]

```
void rti::cds::Logger::service_verbosity (
            rti::config::Verbosity verbosity )  [inline]
```

Sets the verbosity for the log messages generated at the RTI Cloud Discovery **Service** (p. 23) level.

Parameters

verbosity	<<*in*>> The verbosity of the service logs

8.2.2.2 service_verbosity() [2/2]

```
rti::config::Verbosity rti::cds::Logger::service_verbosity ( )  [inline]
```

Getter for the same attribute.

See also

service_verbosity(rti::config::Verbosity) (p. 19)

8.2.2.3 error() [1/2]

```
void rti::cds::Logger::error (
            const std::string & msg )  [inline]
```

Logs a message with EXCEPTION level.

The generated log will be part of logging stream of the running service if the service verbosity contains EXCEPTION messages. The result log message may include additional information according to the Connext logging configuration, such as Advlog Context, thread ID, line number, etc. Additionally, the result log message will contain a newline character at the end, so the format does not need to contain it.

Parameters

in	*msg*	Message to be logged

References **error()**.

Referenced by **error()**.

8.2.2.4 error() [2/2]

```
void rti::cds::Logger::error (
            const char * msg )  [inline]
```

Overload of **error(const std::string& msg)** (p. 20)

8.2.2.5 warn() [1/2]

```
void rti::cds::Logger::warn (
            const std::string & msg )  [inline]
```

Logs as message with WARNING level.

This operation behaves similar to **error()** (p. 20) except the log level is WARNING.

Parameters

in	*msg*	Message to be logged

See also

rti::cds::Logger::error (p. 20)

References **warn()**.

Referenced by **rti::cds::Service::finalize_globals()**, and **warn()**.

8.2.2.6 warn() [2/2]

```
void rti::cds::Logger::warn (
            const char * msg )  [inline]
```

Overload of **warn(const std::string& msg)** (p. 20)

8.2.2.7 local() [1/2]

```
void rti::cds::Logger::local (
            const std::string & msg )  [inline]
```

Logs as message with WARNING level.

This operation behaves similar to **error()** (p. 20) except the log level is STATUS_LOCAL.

Parameters

in	*msg*	Message to be logged

See also

rti::cds::Logger::error (p. 20)

References **local()**.

Referenced by **local()**.

8.2.2.8 local() [2/2]

```
void rti::cds::Logger::local (
            const char * msg )  [inline]
```

Overload of **local(const std::string& msg)** (p. 21)

8.2.2.9 remote() [1/2]

```
void rti::cds::Logger::remote (
            const std::string & msg )  [inline]
```

Logs a message with WARNING level.

This operation behaves similar to **error()** (p. 20) except the log level is STATUS_REMOTE.

Parameters

in	*msg*	Message to be logged

See also

rti::cds::Logger::error (p. 20)

References **remote()**.

Referenced by **remote()**.

8.2.2.10 remote() [2/2]

```
void rti::cds::Logger::remote (
            const char * msg )  [inline]
```

Overload of **remote(const std::string& msg)** (p. 22)

8.2.2.11 debug() [1/2]

```
void rti::cds::Logger::debug (
            const std::string & msg )  [inline]
```

Logs as message with WARNING level.

This operation behaves similar to **error()** (p. 20) except the log level is STATUS_DEBUG.

Parameters

in	*msg*	Message to be logged

See also

rti::cds::Logger::error (p. 20)

References **debug()**.

Referenced by **debug()**.

8.2.2.12 debug() [2/2]

```
void rti::cds::Logger::debug (
            const char * msg )  [inline]
```

Overload of **debug(const std::string& msg)** (p. 22)

8.3 rti::cds::Service Class Reference

The RTI Cloud Discovery **Service** (p. 23).

`#include <Service.hpp>`

Inherits dds::core::Reference< CloudDiscoveryServiceImpl >.

Public Member Functions

- **Service** (const **ServiceProperty** &property)
  *Creates a RTI Cloud Discovery **Service** (p. 23) instance.*
- void **start** ()
  *Starts RTI Cloud Discovery **Service** (p. 23).*
- void **stop** ()
  *Stops RTI Cloud Discovery **Service** (p. 23).*
- bool **is_started** ()
  *Returns true if the RTI Cloud Discovery **Service** (p. 23) is started.*
- bool **update_rtps_psk_secret_passphrase** (const std::string rtps_psk_secret_passphrase)
  *Sets the dds.sec.crypto.rtps_psk_secret_passphrase property for the RTI Cloud Discovery **Service** (p. 23) instance that is started.*

Static Public Member Functions

- static void **finalize_globals** ()

8.3.1 Detailed Description

The RTI Cloud Discovery **Service** (p. 23).

8.3.2 Constructor & Destructor Documentation

8.3.2.1 Service()

```
rti::cds::Service::Service (
            const  ServiceProperty & property )  [inline], [explicit]
```

Creates a RTI Cloud Discovery **Service** (p. 23) instance.

Parameters

property	<<*in*>> The property to configure RTI Cloud Discovery **Service** (p. 23) instance.

8.3.3 Member Function Documentation

8.3.3.1 start()

```
void rti::cds::Service::start ( )  [inline]
```

Starts RTI Cloud Discovery **Service** (p. 23).

This is a non-blocking operation. RTI Cloud Discovery **Service** (p. 23) will create its own set of threads to perform its tasks.

8.3.3.2 stop()

```
void rti::cds::Service::stop ( )  [inline]
```

Stops RTI Cloud Discovery **Service** (p. 23).

This operation will block until the instance is fully stopped.

8.3.3.3 is_started()

```
bool rti::cds::Service::is_started ( )  [inline]
```

Returns true if the RTI Cloud Discovery **Service** (p. 23) is started.

This operation is useful to inspect if the RTI Cloud Discovery **Service** (p. 23) instance is up and running.

8.3.3.4 update_rtps_psk_secret_passphrase()

```
bool rti::cds::Service::update_rtps_psk_secret_passphrase (
            const std::string rtps_psk_secret_passphrase )  [inline]
```

Sets the dds.sec.crypto.rtps_psk_secret_passphrase property for the RTI Cloud Discovery **Service** (p. 23) instance that is started.

This operation performs an update; if there was no value for the dds.sec.crypto.rtps_psk_secret_passphrase property in the XML configuration when RTI Cloud Discovery **Service** (p. 23) was started, the operation would fail.

Parameters

rtps_psk_secret_passphrase	The new value for the dds.sec.crypto.rtps_psk_secret_passphrase property

8.3.3.5 finalize_globals()

```
static void rti::cds::Service::finalize_globals ( )  [inline], [static]
```

[DEPRECATED] Calling this function at the end of the application is no longer necessary. It will be removed in future versions.

References **rti::cds::Logger::warn()**.

8.4 rti::cds::ServiceProperty Class Reference

Configuration properties for a RTI Cloud Discovery **Service** (p. 23) object.

```
#include <ServiceProperty.hpp>
```

Inherits rti::core::NativeValueType< ServiceProperty >.

Public Member Functions

- **ServiceProperty** ()

  Creates a property object with default settings.
- std::string **cfg_file** () const

  Getter (see setter with the same name)
- **ServiceProperty** & **cfg_file** (const std::string &filename)

  *Path to an RTI Cloud Discovery **Service** (p. 23) configuration file.*
- const std::vector< std::string > & **cfg_strings** () const

  Getter (see setter with the same name)
- **ServiceProperty** & **cfg_strings** (const std::vector< std::string > &the_cfg_strings)

  XML configuration represented as strings.
- std::string **service_name** () const

  Getter (see setter with the same name)
- **ServiceProperty** & **service_name** (const std::string &name)

  *The name of the RTI Cloud Discovery **Service** (p. 23) configuration to run.*
- std::string **application_name** () const

  Getter (see setter with the same name)
- **ServiceProperty** & **application_name** (const std::string &name)

  *Assigns a name to the execution of the RTI Cloud Discovery **Service** (p. 23).*
- bool **enforce_xsd_validation** () const

  Getter (see setter with the same name)
- **ServiceProperty** & **enforce_xsd_validation** (const bool enforce_xsd_validation)

  Controls whether the service applies XSD validation to the loaded configuration.
- bool **skip_default_files** () const

  Getter (see setter with the same name)
- **ServiceProperty** & **skip_default_files** (const bool &skip)

  *Set it to true to avoid loading the standard files usually loaded by RTI Cloud Discovery **Service** (p. 23).*
- std::vector< **TransportUnicastProperty** > **transport_selection** ()

  Getter (see setter with the same name)
- **ServiceProperty** & **transport_selection** (const std::vector< **TransportUnicastProperty** > &transport_↩ selection)

  Specifies the transport and receive ports used to receive and send discovery traffic. Specifically, a receive resource is uniquely specified by a transport class-receive port pair. Overrides XML configuration if specified.
- **DomainListProperty** **domain_list** ()

  Getter (see setter with the same name)
- **ServiceProperty** & **domain_list** (const **DomainListProperty** &domain_list)

  Specifies the set of domain IDs the service will accept announcements from.
- bool **enable_administration** () const

  Getter (see setter with the same name)
- **ServiceProperty** & **enable_administration** (const bool enable)

  Set this to true to enable remote administration or false to disable it.
- int **administration_domain_id** () const

  Getter (see setter with the same name)
- **ServiceProperty** & **administration_domain_id** (const int domain_id)

  *If **ServiceProperty::enable_administration** (p. 31) is true, this is the domain ID to use for remote administration.*
- bool **enable_monitoring** () const

  Getter (see setter with the same name)

- **ServiceProperty** & **enable_monitoring** (const bool enable)

  Set this to true to enable monitoring or false to disable it.
- int **monitoring_domain_id** () const

  Getter (see setter with the same name)
- **ServiceProperty** & **monitoring_domain_id** (const int domain_id)

  *If **ServiceProperty::enable_monitoring** (p. 32) is true, this is the domain ID to use for publishing monitoring topics.*
- std::string **license_file_name** () const

  Getter (see setter with the same name)
- **ServiceProperty** & **license_file_name** (const std::string &file_name)

  *Path to an RTI Cloud Discovery **Service** (p. 23) license file.*
- std::map< std::string, std::string > **user_environment** () const

  Getter (see setter with the same name)
- **ServiceProperty** & **user_environment** (const std::map< std::string, std::string > &user_environment)

  Dictionary of user variables.

8.4.1 Detailed Description

Configuration properties for a RTI Cloud Discovery **Service** (p. 23) object.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 ServiceProperty()

```
rti::cds::ServiceProperty::ServiceProperty ( )  [inline]
```

Creates a property object with default settings.

8.4.3 Member Function Documentation

8.4.3.1 cfg_file() [1/2]

```
std::string rti::cds::ServiceProperty::cfg_file ( ) const  [inline]
```

Getter (see setter with the same name)

Referenced by **cfg_file()**.

8.4.3.2 cfg_file() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::cfg_file (
            const std::string & filename )  [inline]
```

Path to an RTI Cloud Discovery **Service** (p. 23) configuration file.

[default] Empty string.

References **cfg_file()**.

8.4.3.3 cfg_strings() [1/2]

```
const std::vector< std::string > & rti::cds::ServiceProperty::cfg_strings ( ) const  [inline]
```

Getter (see setter with the same name)

Referenced by **cfg_strings()**.

8.4.3.4 cfg_strings() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::cfg_strings (
            const std::vector< std::string > & the_cfg_strings )  [inline]
```

XML configuration represented as strings.

An array of strings that altogether make up an XML document to configure RTI Cloud Discovery **Service** (p. 23). This parameter is used only if **ServiceProperty::cfg_file** (p. 27) is empty.

The reason for using an array instead of one single string is to get around the limited size of literal strings. In general, if you create the XML string dynamically the vector needs only one element.

[default] Empty.

References **cfg_strings()**.

8.4.3.5 service_name() [1/2]

```
std::string rti::cds::ServiceProperty::service_name ( ) const  [inline]
```

Getter (see setter with the same name)

Referenced by **service_name()**.

8.4.3.6 service_name() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::service_name (
            const std::string & name )  [inline]
```

The name of the RTI Cloud Discovery **Service** (p. 23) configuration to run.

This is the name used to find the <cloud_discovery_service> XML tag in the configuration file; this is the default name that will be used to refer to this execution in remote administration and monitoring if application_name isn't specified.

[default] Empty string.

References **service_name()**.

8.4.3.7 application_name() [1/2]

```
std::string rti::cds::ServiceProperty::application_name ( ) const  [inline]
```

Getter (see setter with the same name)

Referenced by **application_name()**.

8.4.3.8 application_name() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::application_name (
            const std::string & name )  [inline]
```

Assigns a name to the execution of the RTI Cloud Discovery **Service** (p. 23).

Remote commands and status information will refer to the RTI Cloud Discovery **Service** (p. 23) using this name.

[default] service_name if this value is not set.

References **application_name()**.

8.4.3.9 enforce_xsd_validation() [1/2]

```
bool rti::cds::ServiceProperty::enforce_xsd_validation ( ) const  [inline]
```

Getter (see setter with the same name)

References **enforce_xsd_validation()**.

Referenced by **enforce_xsd_validation()**.

8.4.3.10 enforce_xsd_validation() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::enforce_xsd_validation (
            const bool enforce_xsd_validation )  [inline]
```

Controls whether the service applies XSD validation to the loaded configuration.

[default] true

References **enforce_xsd_validation()**.

8.4.3.11 skip_default_files() [1/2]

```
bool rti::cds::ServiceProperty::skip_default_files ( ) const  [inline]
```

Getter (see setter with the same name)

References **skip_default_files()**.

Referenced by **skip_default_files()**.

8.4.3.12 skip_default_files() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::skip_default_files (
            const bool & skip )  [inline]
```

Set it to true to avoid loading the standard files usually loaded by RTI Cloud Discovery **Service** (p. 23).

Only the configuration in **ServiceProperty::cfg_file** (p. 27) or **ServiceProperty::cfg_strings** (p. 28) will be loaded.

[default] false

References **skip_default_files()**.

8.4.3.13 transport_selection() [1/2]

```
std::vector< TransportUnicastProperty > rti::cds::ServiceProperty::transport_selection ( )  [inline]
```

Getter (see setter with the same name)

References **transport_selection()**.

Referenced by **transport_selection()**.

8.4.3.14 transport_selection() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::transport_selection (
            const std::vector< TransportUnicastProperty > & transport_selection ) [inline]
```

Specifies the transport and receive ports used to receive and send discovery traffic. Specifically, a receive resource is uniquely specified by a transport class-receive port pair. Overrides XML configuration if specified.

[default] Unspecified. Use value from XML configuration.

References **transport_selection()**.

8.4.3.15 domain_list() [1/2]

```
DomainListProperty rti::cds::ServiceProperty::domain_list ( ) [inline]
```

Getter (see setter with the same name)

References **domain_list()**.

Referenced by **domain_list()**.

8.4.3.16 domain_list() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::domain_list (
            const DomainListProperty & domain_list ) [inline]
```

Specifies the set of domain IDs the service will accept announcements from.

[default] Unspecified. Use value from XML configuration.

References **domain_list()**.

8.4.3.17 enable_administration() [1/2]

```
bool rti::cds::ServiceProperty::enable_administration ( ) const [inline]
```

Getter (see setter with the same name)

References **enable_administration()**.

Referenced by **enable_administration()**.

8.4.3.18 enable_administration() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::enable_administration (
            const bool enable )  [inline]
```

Set this to true to enable remote administration or false to disable it.

[default] false

References **enable_administration()**.

8.4.3.19 administration_domain_id() [1/2]

```
int rti::cds::ServiceProperty::administration_domain_id ( ) const  [inline]
```

Getter (see setter with the same name)

Referenced by **administration_domain_id()**.

8.4.3.20 administration_domain_id() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::administration_domain_id (
            const int domain_id )  [inline]
```

If **ServiceProperty::enable_administration** (p. 31) is true, this is the domain ID to use for remote administration.

Takes precedence over the XML configuration. If **ServiceProperty::enable_administration** (p. 31) is false, this value is not used even if remote administration is enabled in the XML configuration.

[default] 0

References **administration_domain_id()**.

8.4.3.21 enable_monitoring() [1/2]

```
bool rti::cds::ServiceProperty::enable_monitoring ( ) const  [inline]
```

Getter (see setter with the same name)

References **enable_monitoring()**.

Referenced by **enable_monitoring()**.

8.4.3.22 enable_monitoring() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::enable_monitoring (
            const bool enable )  [inline]
```

Set this to true to enable monitoring or false to disable it.

[default] false

References **enable_monitoring()**.

8.4.3.23 monitoring_domain_id() [1/2]

```
int rti::cds::ServiceProperty::monitoring_domain_id ( ) const  [inline]
```

Getter (see setter with the same name)

Referenced by **monitoring_domain_id()**.

8.4.3.24 monitoring_domain_id() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::monitoring_domain_id (
            const int domain_id )  [inline]
```

If **ServiceProperty::enable_monitoring** (p. 32) is true, this is the domain ID to use for publishing monitoring topics.

Takes precedence over the XML configuration. If **ServiceProperty::enable_monitoring** (p. 32) is false, this value is not used even if monitoring is enabled in the XML configuration.

[default] 0

References **monitoring_domain_id()**.

8.4.3.25 license_file_name() [1/2]

```
std::string rti::cds::ServiceProperty::license_file_name ( ) const  [inline]
```

Getter (see setter with the same name)

Referenced by **license_file_name()**.

8.4.3.26 license_file_name() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::license_file_name (
            const std::string & file_name )  [inline]
```

Path to an RTI Cloud Discovery **Service** (p. 23) license file.

If not empty, this file is checked for a valid license; otherwise, default location will be used. This parameter is only used if your installation requires a license file.

[default] empty string.

References **license_file_name()**.

8.4.3.27 user_environment() [1/2]

```
std::map< std::string, std::string > rti::cds::ServiceProperty::user_environment ( ) const  [inline]
```

Getter (see setter with the same name)

References **user_environment()**.

Referenced by **user_environment()**.

8.4.3.28 user_environment() [2/2]

```
ServiceProperty & rti::cds::ServiceProperty::user_environment (
            const std::map< std::string, std::string > & user_environment )  [inline]
```

Dictionary of user variables.

The dictionary provides a parallel way to expand XML configuration variables in the form , when they are not defined in the environment.

[default] empty

References **user_environment()**.

8.5 rti::cds::TransportUnicastProperty Class Reference

Specifies the alias name of the transport to be used by RTI Cloud Discovery **Service** (p. 23) for sending or receiving announcements. It also indicates the port where incoming participant announcements are received. Overrides XML configuration if specified.

```
#include <ServiceProperty.hpp>
```

Public Member Functions

- void **alias** (std::string alias)

  Transport alias name.
- std::string & **alias** ()

  Getter (see setter with the same name)
- void **receive_port** (int receive_port)

  Receive port.
- int **receive_port** ()

  Getter (see setter with the same name)

8.5.1 Detailed Description

Specifies the alias name of the transport to be used by RTI Cloud Discovery **Service** (p. 23) for sending or receiving announcements. It also indicates the port where incoming participant announcements are received. Overrides XML configuration if specified.

Refer to `RTI Cloud Discovery Service Available Transports` to check the list of aliases for the preconfigured transports.

8.5.2 Member Function Documentation

8.5.2.1 alias() [1/2]

```
void rti::cds::TransportUnicastProperty::alias (
            std::string alias )  [inline]
```

Transport alias name.

References **alias()**.

8.5.2.2 alias() [2/2]

```
std::string & rti::cds::TransportUnicastProperty::alias ( )  [inline]
```

Getter (see setter with the same name)

Referenced by **alias()**.

8.5.2.3 receive_port() [1/2]

```
void rti::cds::TransportUnicastProperty::receive_port (
            int receive_port )  [inline]
```

Receive port.

References **receive_port()**.

8.5.2.4 receive_port() [2/2]

```
int rti::cds::TransportUnicastProperty::receive_port ( )  [inline]
```

Getter (see setter with the same name)

Referenced by **receive_port()**.

Chapter 9

File Documentation

9.1 CloudDiscoveryServiceImpl.hpp

```
1
10 #ifndef RTI_CDS_CLOUD_DISCOVERY_SERVICE_IMPL_HPP_
11 #define RTI_CDS_CLOUD_DISCOVERY_SERVICE_IMPL_HPP_
12
13 #include <dds/core/refmacros.hpp>
14 #include <rti/core/detail/SelfReference.hpp>
15 #include <rti/core/Exception.hpp>
16 #include "clouddiscoveryservice/clouddiscoveryservice_service.h"
17
18 #include <rti/cds/ServiceProperty.hpp>
19
20 namespace rti { namespace cds {
21
22 class CloudDiscoveryServiceImpl
23         : public rti::core::detail::RetainableType<CloudDiscoveryServiceImpl>
24 {
25 public:
26     explicit CloudDiscoveryServiceImpl(const ServiceProperty& property)
27     {
28         native_ = RTI_CDS_Service_new(&property.native());
29         rti::core::check_create_entity(native_, "CloudDiscoveryService");
30     }
31
32     CloudDiscoveryServiceImpl(const RTI_CDS_Property& property) : native_(NULL)
33     {
34         native_ = RTI_CDS_Service_new(&property);
35         rti::core::check_create_entity(native_, "CloudDiscoveryService");
36     }
37
38     CloudDiscoveryServiceImpl(RTI_CDS_Service *native) : native_(native)
39     {
40         rti::core::check_create_entity(native_, "CloudDiscoveryService");
41     }
42
43     ~CloudDiscoveryServiceImpl()
44     {
45         RTI_CDS_Service_delete(native_);
46     }
47
48     void start()
49     {
50         if (!RTI_CDS_Service_start(native_)) {
51             throw dds::core::Error("Error starting CloudDiscoveryService");
52         }
53     }
54
55     void stop()
56     {
57         if (!RTI_CDS_Service_stop(native_)) {
58             throw dds::core::Error("Error stopping CloudDiscoveryService");
59         }
```

```
60      }
61
62      bool is_started()
63      {
64          return RTI_CDS_Service_is_started(native_);
65      }
66
67      bool update_rtps_psk_secret_passphrase(
68              const std::string rtps_psk_secret_passphrase)
69      {
70          return RTI_CDS_Service_update_rtps_psk_secret_passphrase(
71                  native_,
72                  rtps_psk_secret_passphrase.c_str());
73      }
74
75      RTI_CDS_Service* native() const
76      {
77          return native_;
78      }
79
80  private:
81      RTI_CDS_Service *native_;
82  };
83
84  }}
85
86  #endif // RTI_CDS_CLOUD_DISCOVERY_SERVICE_IMPL_HPP_
```

9.2 Logger.hpp

```
1
10  #ifndef RTI_CDS_LOGGER_HPP_
11  #define RTI_CDS_LOGGER_HPP_
12
13  #include <dds/core/Reference.hpp>
14  #include <rti/config/Logger.hpp>
15
16  #include "clouddiscoveryservice/clouddiscoveryservice_service.h"
17  #include "clouddiscoveryservice/clouddiscoveryservice_log.h"
18
19  namespace rti { namespace cds {
20
33  class Logger
34  {
35  public:
36
37      typedef rti::config::Verbosity Verbosity;
38      typedef rti::config::LogCategory LogCategory;
39      typedef rti::config::PrintFormat PrintFormat;
40
41      static Logger& instance()
42      {
43          static Logger singleton;
44          return singleton;
45      }
46
54      void service_verbosity(rti::config::Verbosity verbosity)
55      {
56          NDDS_Config_Logger_set_verbosity_by_service(
57                  NDDS_Config_Logger_get_instance(),
58                  NDDS_CONFIG_LOG_SERVICE_CLOUD_DISCOVERY,
59                  static_cast<NDDS_Config_LogVerbosity>(verbosity.underlying()));
60      }
61
67       rti::config::Verbosity service_verbosity()
68      {
69          RTILogBitmap instrumentation_mask =
70                  NDDS_Config_Logger_get_verbosity_by_service(
71                          NDDS_Config_Logger_get_instance(),
72                          NDDS_CONFIG_LOG_SERVICE_CLOUD_DISCOVERY);
73
74          return static_cast<rti::config::Verbosity::type>(instrumentation_mask);
75      }
76
89      void error(const std::string& msg)
90      {
91          this->error(msg.c_str());
```

```
92      }
93
97      void error(const char *msg)
98      {
99          this->message(rti::config::LogLevel::EXCEPTION, msg);
100     }
101
113     void warn(const std::string& msg)
114     {
115         this->warn(msg.c_str());
116     }
117
121     void warn(const char *msg)
122     {
123         this->message(rti::config::LogLevel::WARNING, msg);
124     }
125
137     void local(const std::string& msg)
138     {
139         this->local(msg.c_str());
140     }
141
145     void local(const char *msg)
146     {
147         this->message(rti::config::LogLevel::STATUS_LOCAL, msg);
148     }
149
161     void remote(const std::string& msg)
162     {
163         this->remote(msg.c_str());
164     }
165
169     void remote(const char *msg)
170     {
171         this->message(rti::config::LogLevel::STATUS_REMOTE, msg);
172     }
173
185     void debug(const std::string& msg)
186     {
187         this->debug(msg.c_str());
188     }
189
193     void debug(const char *msg)
194     {
195         this->message(rti::config::LogLevel::STATUS_ALL, msg);
196     }
197
198     virtual ~Logger()
199     {
200     }
201
202 private:
203
204     void message(const rti::config::LogLevel& level, const char* msg)
205     {
206         RTI_CDS_Logger_log(
207                 static_cast<NDDS_Config_LogLevel>(level.underlying()),
208                 "%s",
209                 msg);
210     }
211
212     Logger()
213     {
214     }
215
216     // Disable copy
217     Logger(const Logger&);
218     Logger& operator=(const Logger&);
219 };
220
221 }}
222
223 #endif // RTI_CDS_LOGGER_HPP_
```

9.3 PropertySet.hpp

```
1 #ifndef RTI_CDS_PROPERTYSET_HPP
```

```
2 #define RTI_CDS_PROPERTYSET_HPP
3
4 #include <rti/routing/PropertySet.hpp>
5
6 namespace rti { namespace recording {
7
8 typedef rti::routing::PropertySet PropertySet;
9
10 }}
11
12 #endif /* RTI_CDS_PROPERTYSET_HPP */
13
14
```

9.4 Service.hpp

```
1
10 #ifndef RTI_CDS_CLOUD_DISCOVERY_SERVICE_HPP_
11 #define RTI_CDS_CLOUD_DISCOVERY_SERVICE_HPP_
12
13 #include <dds/core/Reference.hpp>
14 #include <rti/config/Logger.hpp>
15
16 #include <rti/cds/Logger.hpp>
17 #include <rti/cds/ServiceProperty.hpp>
18 #include <rti/cds/detail/CloudDiscoveryServiceImpl.hpp>
19
20 #include "routingservice/routingservice_service.h"
21 #include "clouddiscoveryservice/clouddiscoveryservice_log.h"
22
23 namespace rti { namespace cds {
24
25 /*e \defgroup RTI_CDS_ServiceLibModule Library API
26  *
27  * @brief @product can be deployed as a native library linked into your
28  * application in select architectures.
29  *
30  * This API allows you to create, configure and start @product instances from
31  * your application.
32  *
33  * The following code shows the typical use of the API:
34  *
35  * \code
36  *
37  * int main ()
38  * {
39  *     rti::cds::Service service(
40  *             rti::cds::ServiceProperty()
41  *             .cfg_file("MyCloudDiscoveryService.xml")
42  *             .service_name("MyCloudDiscoveryService"));
43  *     ...
44  *
45  *     service.start();
46  *
47  *     while (keep_running) {
48  *         sleep();
49  *         ...
50  *     }
51  *
52  *     return 0;
53  * }
54  *
55  * \endcode
56  *
57  * Instead of a file, you can use XML strings to configure @product.
58  * See ServiceProperty for more information.
59  * <p>
60  * To build your application you need to link with the @product native library
61  * in <b> &lt;@ndds home&gt;/bin/&lt;architecture&gt;/ </b>.
62  *
63  * An example is provided at
      <b>https://github.com/rticommunity/rticonnextdds-examples/tree/master/examples/cloud_discovery_service/library_api</b>.
64  *
65  * ### Development Requirements
66  *
67  * |                              | Linux Systems                  | Windows Systems               |
68  * | ---------------------------: | :----------------------------: | :---------------------------: |
```

```
69  * | Link Libraries (Compile Time) | librticlouddiscoveryservice.so | rticlouddiscoveryservice.dll  |
70  * | Shared Libraries (Run Time)   | librtiroutingservice.so        | rtiroutingservice.dll         |
71  * | ^                             | librtidlc.so                   | rtidlc.dll                    |
72  * | ^                             | librticonnextmsgc.so           | rticonnextmsgc.dll            |
73  * | ^                             | libnddsmetp.so                 | nddsmetp.dll                  |
74  * | ^                             | librtiapputilsc.so             | rtiapputilsc.dll              |
75  * | ^                             | librtixml2.so                  | rtixml2.dll                   |
76  * | ^                             | libnddscpp2.so                 | nddscpp2.dll                  |
77  * | ^                             | libnddsc.so                    | nddsc.dll                     |
78  * | ^                             | libnddscore.so                 | nddscore.dll                  |
79  * | Static Libraries              | librticlouddiscoveryservicez.a | rticlouddiscoveryservicez.lib |
80  * | ^                             | librtiroutingservicez.a        | rtiroutingservicez.lib        |
81  * | ^                             | librtidlcz.a                   | rtidlcz.lib                   |
82  * | ^                             | librticonnextmsgcz.a           | rticonnextmsgcz.lib           |
83  * | ^                             | libnddsmetpz.a                 | nddsmetpz.lib                 |
84  * | ^                             | librtiapputilscz.a             | rtiapputilscz.lib             |
85  * | ^                             | librtixml2z.a                  | rtixml2z.lib                  |
86  * | ^                             | libnddscpp2z.a                 | nddscpp2z.lib                 |
87  * | ^                             | libnddscz.a                    | nddscz.lib                    |
88  * | ^                             | libnddscorez.a                 | nddscorez.lib                 |
89  * | Headers                       | rti/cds/CloudDiscoveryService.hpp | |
90  *
91  * <b>NOTE:</b> If you are using debug libraries, remember to add the 'd' suffix
92  * to the library name. For shared libraries on macOS systems, the library name
93  * is the same as that for Linux except the extension is '.dylib' instead of '.so'.
94  */
95
101 class Service : public dds::core::Reference<CloudDiscoveryServiceImpl> {
102 public:
103     typedef dds::core::Reference<CloudDiscoveryServiceImpl> Base;
104     OMG_DDS_REF_TYPE_NOTYPENAME(
105             Service,
106             dds::core::Reference,
107             CloudDiscoveryServiceImpl);
108
116     explicit Service(const ServiceProperty& property)
117             : Base(new CloudDiscoveryServiceImpl(property))
118     {
119         this->delegate()->remember_reference(this->delegate());
120     }
121
122     Service(const RTI_CDS_Property& property)
123             : Base(new CloudDiscoveryServiceImpl(property))
124     {
125         this->delegate()->remember_reference(this->delegate());
126     }
127
128     explicit Service(Base::DELEGATE_REF_T reference) : Base(reference)
129     {
130         if (this->delegate()) {
131             this->delegate()->remember_reference(this->delegate());
132         }
133     }
134
141     void start()
142     {
143         this->delegate()->start();
144     }
145
151     void stop()
152     {
153         this->delegate()->stop();
154     }
155
161     bool is_started()
162     {
163         return this->delegate()->is_started();
164     }
165
177     bool update_rtps_psk_secret_passphrase(
178             const std::string rtps_psk_secret_passphrase)
179     {
180         return this->delegate()->update_rtps_psk_secret_passphrase(
181                 rtps_psk_secret_passphrase);
182     }
183
188     static void finalize_globals()
189     {
190         rti::cds::Logger::instance().warn(
191                 "Calling 'rti::cds::Service::finalize_globals()' is no "
192                 "longer necessary and the function will be removed in future "
```

```
193                 "versions");
194     }
195
196 };
197
198 } }
199
200 #endif // RTI_CDS_CLOUD_DISCOVERY_SERVICE_HPP_
```

9.5 ServiceProperty.hpp

```
1
10 #ifndef RTI_CDS_CLOUD_DISCOVERY_SERVICE_PROPERTY_HPP_
11 #define RTI_CDS_CLOUD_DISCOVERY_SERVICE_PROPERTY_HPP_
12
17 #include <dds/core/macros.hpp>
18 #include <rti/core/NativeValueType.hpp>
19
20 #include "clouddiscoveryservice/clouddiscoveryservice_infrastructure.h"
21 #include "clouddiscoveryservice/clouddiscoveryservice_service.h"
22
23 namespace rti { namespace cds {
24
30 class DomainListProperty {
31 public:
32     DomainListProperty() : allow_domain_ids_(""), deny_domain_ids_("")
33     {
34     }
35
36     DomainListProperty(std::string allow_domain_ids, std::string deny_domain_ids)
37     {
38         allow_domain_ids_ = allow_domain_ids;
39         deny_domain_ids_ = deny_domain_ids;
40     }
41
48     void allow_domain_ids(std::string allow_domain_ids)
49     {
50         allow_domain_ids_ = allow_domain_ids;
51     }
52
56     std::string& allow_domain_ids()
57     {
58         return allow_domain_ids_;
59     }
60
67     void deny_domain_ids(std::string deny_domain_ids)
68     {
69         deny_domain_ids_ = deny_domain_ids;
70     }
71
75     std::string& deny_domain_ids()
76     {
77         return deny_domain_ids_;
78     }
79
80 private:
81     std::string allow_domain_ids_;
82     std::string deny_domain_ids_;
83 };
84
94 class TransportUnicastProperty {
95 public:
96     TransportUnicastProperty(std::string alias, int receive_port)
97     {
98         alias_ = alias;
99         receive_port_ = receive_port;
100     }
101
105     void alias(std::string alias)
106     {
107         alias_ = alias;
108     }
109
113     std::string& alias()
114     {
115         return alias_;
116     }
```

```
117
121     void receive_port(int receive_port)
122     {
123         receive_port_ = receive_port;
124     }
125
129     int receive_port()
130     {
131         return receive_port_;
132     }
133
134 private:
135     std::string alias_;
136     int receive_port_;
137 };
138
139 class PropertyAdapter {
140 public:
141     typedef RTI_CDS_Property native_type;
142
143     static void initialize(native_type& native_value)
144     {
145         static const native_type DEFAULT
146                 = RTI_CDS_Property_INITIALIZER;
147         native_value = DEFAULT;
148         // Replicating the defaults from the C Library API as specified in service.ifc
149         native_value.service_verbosity = RTI_LOG_BIT_EXCEPTION;
150         native_value.dds_verbosity = RTI_LOG_BIT_EXCEPTION;
151     }
152
153     static void finalize(native_type& native_value)
154     {
155         if (native_value.cfg_strings != NULL) {
156             RTIOsapiHeap_freeArray(native_value.cfg_strings);
157             native_value.cfg_strings = NULL;
158         }
159
160         RTI_CDS_Property_finalize(&native_value);
161     }
162
163     static void copy(native_type& destination, const native_type& source)
164     {
165         if (RTI_CDS_Property_copy(&destination, &source) == NULL) {
166             throw dds::core::Error("Native RTI_CDS_Property_copy");
167         }
168     }
169
170     static bool equals(const native_type& first, const native_type& second)
171     {
172         RTIOsapiUtility_unusedParameter(first);
173         RTIOsapiUtility_unusedParameter(second);
174
175         throw dds::core::UnsupportedError("Unsupported operation: equals");
176     }
177
178     static void add_transports_from_native(
179             std::vector<TransportUnicastProperty>& transport_selection,
180             const struct RTI_CDS_TransportSelectionProperty *transport_selection_native)
181     {
182         for (int i = 0; i < transport_selection_native->count; i++) {
183             transport_selection.push_back(TransportUnicastProperty(
184                     transport_selection_native->transports[i].alias,
185                     transport_selection_native->transports[i].receive_port));
186         }
187     }
188
189     static void add_transports_to_native(
190             struct RTI_CDS_TransportSelectionProperty *transport_selection_native,
191             std::vector<TransportUnicastProperty>& transport_selection)
192     {
193         transport_selection_native->count = transport_selection.size();
194         for (unsigned int i = 0; i < transport_selection.size(); i++) {
195             RTIOsapiUtility_strcpy(
196                     transport_selection_native->transports[i].alias,
197                     RTI_CDS_TRANSPORT_UNICAST_PROPERTY_ALIAS_LENGTH_MAX,
198                     transport_selection[i].alias().c_str());
199             transport_selection_native->transports[i].receive_port
200                     = transport_selection[i].receive_port();
201         }
202     }
203
```

```
204     static void add_domain_list_from_native(
205             DomainListProperty& domain_ids,
206             const struct RTI_CDS_DomainListProperty *domain_ids_native)
207     {
208         domain_ids.allow_domain_ids(domain_ids_native->allow_domain_ids);
209         domain_ids.deny_domain_ids(domain_ids_native->deny_domain_ids);
210     }
211
212     static void add_domain_list_to_native(
213             struct RTI_CDS_DomainListProperty *domain_ids_native,
214             DomainListProperty& domain_ids)
215     {
216         DDS_String_replace(
217                 &domain_ids_native->allow_domain_ids,
218                 domain_ids.allow_domain_ids().c_str());
219         DDS_String_replace(
220                 &domain_ids_native->deny_domain_ids,
221                 domain_ids.deny_domain_ids().c_str());
222     }
223
224     static void add_properties_from_native(
225             std::map<std::string, std::string>& properties,
226             const RTI_RoutingServiceProperties *native_properties)
227     {
228         for (int i = 0; i < native_properties->count; i++) {
229             properties[native_properties->properties[i].name] =
230                     (const char *) native_properties->properties[i].value;
231         }
232     }
233
234     static void add_properties_to_native(
235             RTI_RoutingServiceProperties *native_properties,
236             const std::map<std::string, std::string>& properties)
237     {
238         RTI_RoutingServiceProperties_finalize(native_properties);
239         if (!RTI_RoutingServiceProperties_initialize(native_properties)) {
240             throw dds::core::Error("failed to initialize native property");
241         }
242
243         std::map<std::string, std::string>::const_iterator it = properties.begin();
244         for (; it != properties.end(); ++it) {
245             if (!RTI_RoutingServiceProperties_add(
246                     native_properties,
247                     it->first.c_str(),
248                     it->second.c_str())) {
249                 throw dds::core::Error("failed to add native property");
250             }
251         }
252     }
253 };
254
255 class ServiceProperty;
256
257 }
258
259 namespace core {
260
261 template<>
262 struct native_type_traits<rti::cds::ServiceProperty> {
263     typedef rti::cds::PropertyAdapter adapter_type;
264     typedef RTI_CDS_Property native_type;
265 };
266
267 }
268
269 namespace cds {
270
276 class ServiceProperty : public rti::core::NativeValueType<ServiceProperty>
277 {
278 public:
279     RTI_NATIVE_VALUE_TYPE_DEFINE_DEFAULT_MOVE_OPERATIONS(ServiceProperty)
280
281     typedef rti::core::NativeValueType<ServiceProperty> Base;
282
286     ServiceProperty(): Base()
287     {
288     }
289
290     ServiceProperty(const struct RTI_CDS_Property &native_property)
291             : Base(native_property)
292     {
```

```
293     }
294
298     std::string cfg_file() const
299     {
300         return native().cfg_file == NULL ? "" : native().cfg_file;
301     }
302
308     ServiceProperty& cfg_file(const std::string& filename)
309     {
310         DDS_String_replace(
311                 &native().cfg_file,
312                 filename.c_str());
313         return *this;
314     }
315
319     const std::vector<std::string>& cfg_strings() const
320     {
321         return cfg_strings_;
322     }
323
338     ServiceProperty& cfg_strings(
339             const std::vector<std::string>& the_cfg_strings)
340     {
341         cfg_strings_ = the_cfg_strings;
342         if (cfg_strings_.size() > 0) {
343             if (!RTIOsapiHeap_reallocateArray(
344                     (char ***) &native().cfg_strings,
345                     cfg_strings_.size(),
346                     char *)) {
347                 throw dds::core::OutOfResourcesError("cfg_strings native array");
348             }
349
350             for (uint32_t i = 0; i < cfg_strings_.size(); i++) {
351                 native().cfg_strings[i] = cfg_strings_[i].c_str();
352             }
353         }
354         native().cfg_strings_count = static_cast<int>(cfg_strings_.size());
355
356         return *this;
357     }
358
362     std::string service_name() const
363     {
364         return native().service_name == NULL ? "" : native().service_name;
365     }
366
377     ServiceProperty& service_name(const std::string& name)
378     {
379         DDS_String_replace(&native().service_name, name.c_str());
380         return *this;
381     }
382
386     std::string application_name() const
387     {
388         return native().application_name == NULL ? "" : native().application_name;
389     }
390
399     ServiceProperty& application_name(const std::string& name)
400     {
401         DDS_String_replace(&native().application_name, name.c_str());
402         return *this;
403     }
404
408     bool enforce_xsd_validation() const
409     {
410         return (native().enforce_xsd_validation == DDS_BOOLEAN_TRUE)
411                 ? true
412                 : false;
413     }
414
421     ServiceProperty& enforce_xsd_validation(const bool enforce_xsd_validation)
422     {
423         native().enforce_xsd_validation = enforce_xsd_validation;
424         return *this;
425     }
426
430     bool skip_default_files() const
431     {
432         return (native().skip_default_files == DDS_BOOLEAN_TRUE)
433                 ? true
434                 : false;
```

```
435     }
436
446     ServiceProperty& skip_default_files(const bool& skip)
447     {
448         native().skip_default_files = skip
449                 ? DDS_BOOLEAN_TRUE
450                 : DDS_BOOLEAN_FALSE;
451         return *this;
452     }
453
457     std::vector<TransportUnicastProperty> transport_selection()
458     {
459         PropertyAdapter::add_transports_from_native(
460                 transport_selection_,
461                 &native().transport_selection);
462
463         return transport_selection_;
464     }
465
474     ServiceProperty& transport_selection(
475             const std::vector<TransportUnicastProperty>& transport_selection)
476     {
477         transport_selection_ = transport_selection;
478         PropertyAdapter::add_transports_to_native(
479                 &native().transport_selection,
480                 transport_selection_);
481
482         return *this;
483     }
484
488     DomainListProperty domain_list()
489     {
490         PropertyAdapter::add_domain_list_from_native(
491                 domain_list_,
492                 &native().domain_list);
493
494         return domain_list_;
495     }
496
503     ServiceProperty& domain_list(
504             const DomainListProperty& domain_list)
505     {
506         domain_list_ = domain_list;
507         PropertyAdapter::add_domain_list_to_native(
508                 &native().domain_list,
509                 domain_list_);
510
511         return *this;
512     }
513
517     bool enable_administration() const
518     {
519         return (native().enable_administration == DDS_BOOLEAN_TRUE)
520                 ? true
521                 : false;
522     }
523
530     ServiceProperty& enable_administration(const bool enable)
531     {
532         native().enable_administration = enable
533                 ? DDS_BOOLEAN_TRUE
534                 : DDS_BOOLEAN_FALSE;
535         return *this;
536     }
537
541     int administration_domain_id() const
542     {
543         return native().administration_domain_id;
544     }
545
556     ServiceProperty& administration_domain_id(const int domain_id)
557     {
558         native().administration_domain_id = domain_id;
559         return *this;
560     }
561
565     bool enable_monitoring() const
566     {
567         return (native().enable_monitoring == DDS_BOOLEAN_TRUE)
568                 ? true
569                 : false;
```

```
570     }
571
577     ServiceProperty& enable_monitoring(const bool enable)
578     {
579         native().enable_monitoring = enable
580                 ? DDS_BOOLEAN_TRUE
581                 : DDS_BOOLEAN_FALSE;
582         return *this;
583     }
584
588     int monitoring_domain_id() const
589     {
590         return native().monitoring_domain_id;
591     }
592
603     ServiceProperty& monitoring_domain_id(const int domain_id)
604     {
605         native().monitoring_domain_id = domain_id;
606         return *this;
607     }
608
612     std::string license_file_name() const
613     {
614         return native().license_file_name == NULL
615                 ? ""
616                 : native().license_file_name;
617     }
618
628     ServiceProperty& license_file_name(const std::string& file_name)
629     {
630         DDS_String_replace(
631                 &native().license_file_name,
632                 file_name.c_str());
633         return *this;
634     }
635
639     std::map<std::string, std::string> user_environment() const
640     {
641         std::map<std::string, std::string> properties;
642
643         PropertyAdapter::add_properties_from_native(
644                 properties,
645                 &native().user_environment);
646
647         return properties;
648     }
649
659     ServiceProperty& user_environment(
660             const std::map<std::string, std::string>& user_environment)
661     {
662         PropertyAdapter::add_properties_to_native(
663                 &native().user_environment,
664                 user_environment);
665
666         return *this;
667     }
668
669 private:
670     std::vector<std::string> cfg_strings_;
671     std::vector<TransportUnicastProperty> transport_selection_;
672     DomainListProperty domain_list_;
673
674 };
675
676 }}
677
678 #endif // RTI_CDS_CLOUD_DISCOVERY_SERVICE_PROPERTY_HPP_
```

Index

administration_domain_id
    rti::cds::ServiceProperty, 32
alias
    rti::cds::TransportUnicastProperty, 35
allow_domain_ids
    rti::cds::DomainListProperty, 17, 18
application_name
    rti::cds::ServiceProperty, 29

cfg_file
    rti::cds::ServiceProperty, 27
cfg_strings
    rti::cds::ServiceProperty, 28
CloudDiscoveryServiceImpl.hpp, 37

debug
    rti::cds::Logger, 22, 23
deny_domain_ids
    rti::cds::DomainListProperty, 18
domain_list
    rti::cds::ServiceProperty, 31

enable_administration
    rti::cds::ServiceProperty, 31
enable_monitoring
    rti::cds::ServiceProperty, 32
enforce_xsd_validation
    rti::cds::ServiceProperty, 29
error
    rti::cds::Logger, 20

finalize_globals
    rti::cds::Service, 25

is_started
    rti::cds::Service, 24

Library API, 11
license_file_name
    rti::cds::ServiceProperty, 33
local
    rti::cds::Logger, 21, 22
Logger.hpp, 38
Logging, 11

monitoring_domain_id
    rti::cds::ServiceProperty, 33

PropertySet.hpp, 39

receive_port
    rti::cds::TransportUnicastProperty, 35, 36
remote
    rti::cds::Logger, 22
rti, 15
rti::cds::DomainListProperty, 17
    allow_domain_ids, 17, 18
    deny_domain_ids, 18
rti::cds::Logger, 19
    debug, 22, 23
    error, 20
    local, 21, 22
    remote, 22
    service_verbosity, 19, 20
    warn, 20, 21
rti::cds::Service, 23
    finalize_globals, 25
    is_started, 24
    Service, 24
    start, 24
    stop, 24
    update_rtps_psk_secret_passphrase, 25
rti::cds::ServiceProperty, 25
    administration_domain_id, 32
    application_name, 29
    cfg_file, 27
    cfg_strings, 28
    domain_list, 31
    enable_administration, 31
    enable_monitoring, 32
    enforce_xsd_validation, 29
    license_file_name, 33
    monitoring_domain_id, 33
    service_name, 28
    ServiceProperty, 27
    skip_default_files, 30
    transport_selection, 30
    user_environment, 34
rti::cds::TransportUnicastProperty, 34
    alias, 35
    receive_port, 35, 36

Service
    rti::cds::Service, 24

Service.hpp, 40
service_name
    rti::cds::ServiceProperty, 28
service_verbosity
    rti::cds::Logger, 19, 20
ServiceProperty
    rti::cds::ServiceProperty, 27
ServiceProperty.hpp, 42
skip_default_files
    rti::cds::ServiceProperty, 30
start
    rti::cds::Service, 24
stop
    rti::cds::Service, 24

transport_selection
    rti::cds::ServiceProperty, 30

update_rtps_psk_secret_passphrase
    rti::cds::Service, 25
user_environment
    rti::cds::ServiceProperty, 34

warn
    rti::cds::Logger, 20, 21